PROJECT PROGRESS REPORT IV

For The Project Of

LOCALLY CONNECTED ADAPTIVE GABOR FILTER FOR REAL-TIME MOTION COMPENSATION

For the Period from July 20th of 1994
to October 19th of 1994

This Report Is Submitted to ONR

October 19th, 1994

Principal Investigator: Professor Hua Li
Computer Science Department, College of Engineering
Texas Tech University, Lubbock, TX 79409
Telephone: (806) 742-3513, E-mail: xdhua@ttacs1.ttu.edu

Administrative Point of Contact
Kathleen Harris, Ph.D., Director of
The Office of Research Services
Texas Tech University
Lubbock, Tx 79409
Phone: (806) 742-3884

PROJECT PROGRESS REPORT IV

For The Project Of

LOCALITY CONNECTED ADAPTIVE GABOR FILTER FOR REAL-TIME MOTION COMPENSATION

For the Period from July 20th of 1994 to October 19th of 1994

Accesion For	
NTIS CRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By A28[illegible]544	
Distribution /	
Availability Codes	
Dist	Avail and / or Special
A-1	

This Report Is Submitted to ONR

October 19th, 1994

Principal Investigator: Professor Hua Li
Computer Science Department, College of Engineering
Texas Tech University, Lubbock, TX 79409
Telephone: (806) 742-3513, E-mail: xdhua@ttacs1.ttu.edu

Administrative Point of Contact
Kathleen Harris, Ph.D., Director of
The Office of Research Services
Texas Tech University
Lubbock, Tx 79409
Phone: (806) 742-3884

Table of Contents

The Progress Report

This report is the forth quartly report for the project of "*locally connected adaptive Gabor filter for real-time motion compensation,*" with grant number N00014-94-1-0077, which has been in the process since October 20th of 1993 and has been conducted under the supervision of the principal investigator, Professor Hua Li of Texas Tech University.

As at the end of the first year of this three-year project, we have been making progress towards the goals of this research as planned in the project proposal, 1(b) on page 21. In particular, we have been accomplished the following work as itemized below:

1. We have conducted extensive research work on the design and simulation of electronic analog circuits as basic building blocks for the VLSI implementation. This phase of the work is a little bit ahead of the schedule (about 1 month) than we originally planned in the proposal. This hardware design phase concurrent with the algorithm analysis and verification provided coherent work and ensured the quality of the analog VLSI design. The work in VLSI implementation at this stage includes

   (a) Design one of the most essential building blocks, a video frequency opAmp. With extensive SPICE simulations using the device model provided from MOSIS actual fabrication run, we have completed the design. The characteristics of the OpAmp to be used to build two-dimensional convolution unit is given in a research memo and was included in the 2nd quart report as Appendix II.

   (b) Design analog multiplier, which is to be used for building two-dimensional analog convolution unit. Extensive SPICE simulation was performed. The preliminary simulation data looks very positive and the circuit layout design has been completed.

   (c) Post-layout SPICE simulation is under way to check the actual VLSI implementation. The experimental result now is under analysis and will be documented and reported accordingly. Some results have been documented and given in Appendix I of this report.

2. In order to benchmark the VLSI chips, a hardware prototype board is under design and construction. This board will be used to compare the performance of digital approach vs. analog approach, analog approach based on the standard

off-the shelf components vs. analog customer-design VLSI approach. The board is now operating and it is under calibration. The board is designed with a 486 machine as a host. The prototype demonstration is prepared for ONR.

A software package has been developed to implement the Gabor motion detection algorithm. This software consists of utility functions, algorithm modules, and test pattern generators for the experiments and verification of the *spatial* and *temporal selectivity*. A software program manual now has been produced. Fully tested, documented programs are provided together with the manual.
This software package consists of the following programs:

1. Programs for algorithm implementation, which include:
   - (a) Real Gabor kernel with user-selectable kernel size, spatial frequency, and orientation frequency;
   - (b) Imaginary Gabor kernel with user-selectable kernel size, spatial frequency, and orientation frequency;
   - (c) Derivatives (with respect to x and y) of the real Gabor kernel with user-selectable kernel size, spatial frequency, and orientation frequency;
   - (d) Derivatives (with respect to x and y) of the imaginary Gabor kernel with user-selectable kernel size, spatial frequency, and orientation frequency;
   - (e) Image convolution program with user-selectable kernels;
   - (f) Least square estimation algorithm for solving an over-determined linear system, which is needed at the last phase for optical flow computation.
2. Utility programs which consist of
   - (a) A set of programs to creat test image patterns based on *virtual reality technique*. These programs allow user to define the observer's position (camera position), the fixation direction of the camera, and the orientation of the projection plane (virtual film.) They can be utilized by the user to define objects in three-dimensional world-coordinated space.
   - (b) A program to creat animated images. This program will produce each individual frame of images, and generate an animated image sequence. These two programs will be particularly needed to test and verify the optical flow computation based on the created known motion of the objects and the observer.

(c) A short program (in MATLAB format) to creat postscript files. This will produce the hardcopy of images, and computation result.

Based on the theoretical analysis, the optical flow was computed for several artificially generated test patterns. These test patterns are designed to test the concept of *spatial* and *orientation selectivity*. The patterns were generated by using *virtual reality* technique based on *three-dimensional computer graphics*. As described in the second quartly report, they include stationary observer while object is moving, and stationary object while observer is moving, as well as both observer and object moving in a known fashion. In order to further test the computation, we have modified the resolution and color depth of each pixel to make them all 120-by-100 in resolution and 8-bit color per pixel. A set of floppy disks are now prepared for the demonstration purpose. This set of floppy disks are now included in this report to ONR. Included in this set of disks are image sequences generated under known conditions as benchmark for the future testing of VLSI implementation, in particular, they include the following:

1. An observer (camera) is stationary, but the viewing object (a sphere with radius equal to 25 units) is moving along y-axis 5 units per frame.
2. The observer is moving along y-axis 5 units per frame while the viewing object is stationary.
3. Both the observer and the viewing object are moving 5 units along y-axis and z-axis respectively.

As briefly summaried here, we have been making progress as planned. In the next phase, the second year of the project, we will produce the fabricated analog VLSI chips for testing and evaluation.

The End.

ANALOG CONVOLUTION UNIT FOR REAL TIME IMAGE PROCESSING

Laszlo Moldovan and Hua Li
Department of Computer Science
College of Engineering, Texas Tech University
Lubbock, TX 79409
E-mail: xdhua@ttacs1.ttu.edu

***Abstract*— The design of an analog convolution unit for real time image processing is presented in this paper with the emphasis on the design of the basic building block: CMOS four quadrant multiplier. This CMOS multiplier operates with ±5V and has single ended voltage inputs making it very easy to use for real time image processing. This basic building block is used to build a 5x5 array of convolution unit. The MAGIC layout of this circuit, ready for double poly CMOS analog fabrication is also presented.**

I. INTRODUCTION

The design of an analog convolution unit for real time image processing is given in this memo. The unit consists of several different kind basic building blocks including an analog multiplier. An analog multiplier can be easily built in bipolar technology and they have been successfully used for years [1]. The problem when trying to build an analog multiplier using CMOS devices is that the output current of the source-coupled differential pair (M5 and M6 in Figure 1) depends nonlinearly on the bias current sinked by M7 and the input voltage. Therefore, the linear range of the input voltages is very limited and very hard to compensate. Given the need of using CMOS technology over bipolar technology, we have designed the basic building block and the unit by using active attenuator technique. This design provides much better linear operating range of the CMOS four quadrant multiplier.

II. PRINCIPLE OF OPERATION OF ANALOG MULTIPLIER

The complete circuit of the multiplier is shown in Figure 1. The core of this circuit consists of the CMOS version of the standard Gilbert cell presented in [1]. As above mentioned, the problem of this circuit is the reduced voltage input swing range. In order to overcome this problem, the input voltages are connected to the Gilbert cell through active attenuators. The output signal of the circuit is a differential current, which can be easily converted to a differential voltage with two load resistors connected to Vdd. The Gilbert cell and the active attenuators are described as follows.

III. GILBERT CELL

Devices M1, M2, M3, M4, M5, M6 and the current sink M7 form a CMOS version of the Gilbert cell. All transistors operate in the saturation region and the transconductance parameters of M1..M4 and M5, M6 are matched and equal to K_1 and K_2 respectively. Therefore, the ideal square-law equation can be applied. The output currents of this circuit are I_{o1} and I_{o2}, given by:

$$I_{o1} = -(I_{d1} + I_{d3}) \tag{1}$$

and

$$I_{o2} = -(I_{d2} + I_{d4}) \tag{2}$$

Thus, the output differential current, $I_{od} = I_{o2} - I_{o1}$ is given by:

$$I_{od} = \sqrt{2K_1}\,V_x'[\sqrt{I_{d5}}\sqrt{1 - \frac{K_1 V_x'^2}{2I_{d5}}} - \sqrt{I_{d6}}\sqrt{1 - \frac{K_1 V_x'^2}{2I_{d6}}}] \tag{3}$$

where $V_{GS1} \equiv V_{GS4} = V_x'$. If terms $\frac{K_1 V_x'^2}{2I_{d5}}$ and $\frac{K_1 V_x'^2}{2I_{d6}}$ are much smaller than 1, it follows that I_{od} depends linearly on V_x' and is given by:

$$I_{od} \simeq \sqrt{2K_1}(\sqrt{I_{d5}} - \sqrt{I_{d6}})V_x' \tag{4}$$

Also, I_{d5} and I_{d6} can be shown to be in the following relationship with $V_{GS2} \equiv V_{GS3} = V_y'$:

$$V_y' \simeq \frac{1}{\sqrt{2K_2}}(\sqrt{I_{d5}} - \sqrt{I_{d6}})V_y' \tag{5}$$

Substituting (5) into (4), we have the output differential current:

$$I_{od} = \sqrt{2K_1K_2}\,V_x'V_y' \tag{6}$$

which is the characteristic of an analog multiplier. But this equation was obtained under the assumption that V_x' and V_y' are kept very small which may not be satisfied. To solve this problem, these two voltages were collected at the outputs of active attenuators.

IV. ACTIVE ATTENUATOR

The active attenuators (one for the VX input another for the VY input) were built with transistors M8, M9 and respectively M11, M12. In order to couple the attenuators with the two inputs, a voltage shift was obtained with the source followers M10, M18 and M13, M19. A voltage (VX or VY) applied to these attenuators plus shifters will be reduced by a factor of 10, as resulted from PSPICE simulations. In other words, the circuit will reduce the slope of a linearly varying voltage between -1V and +1V from 1 to 0.1.

V. MULTIPLIER UNIT

With the attenuators attached, the transfer characteristic of the multiplier unit is described by the equation:

$$I_{od} = I_{o1} - I_{o2} \simeq \sqrt{2K_1K_2}\, m^2 V_x V_y \qquad (7)$$

where m is the attenuation factor of the active attenuators (10.9). So far the output was a differential current. If we want to convert it into a differential voltage, we need to connect two matched load resistors between the outputs of the Gilbert cell and Vdd. Thus the output voltage is given by:

$$V_o = R_L I_{od} \simeq [\sqrt{2K_1K_2}R_L]\, m^2 V_x V_y \qquad (8)$$

In order to be able to calibrate the circuit for different fabrication parameters (which cannot be known precisely in advance) and to compensate for the offset voltage, the attenuators are biased externally on the gates of M14 and M15. As loads, RL1 and RL2 were chosen to be 1KΩ. The sizes of the transistors are given in the following table:

TABLE I
Device sizes for multiplier unit
size W/L=μm/μm

M1	80/4	M9	2/8
M2	80/4	M10	2/3
M3	80/4	M11	2/3
M4	80/4	M12	2/8
M5	80/4	M13	2/3
M6	80/4	M14	16/2
M7	80/4	M15	16/2
M8	2/3		

VI. EXPERIMENTAL RESULTS

First, the circuit was tested using PSPICE. The transfer characteristics for a -1V to 1V dc sweep on VX and VY inputs are shown in Figures 2 and 3. The PSPICE file of the circuit is contained in APPENDIX A. A Bode plot in Figure 4 shows that the circuit has a -3dB bandwidth of 24.4 MHz. Following the preliminary simulations, the layout of this circuit was designed using MAGIC and then its circuit was extracted for PSPICE simulation to verify the layout design. The PSPICE simulations for the extracted circuit gave the transfer characteristics as shown in Figures 5 and 6 and the Bode plot in Figure 7. The MAGIC layout of a single multiplier cell can be seen in Figure 8. The -3 dB bandwidth in this case is 19 MHz. Nonlinearity tests were performed on the analog multiplier for both inputs. First, a ±1V ramp signal was input on VY, VX was made succesively 1V and -1V and the output voltage was multiplied with a constant factor to make Vout/VY=1. Then VY was subtracted from Vout and the result represented the absolute nonlinearity error. From this value, the relative nonlinearity error was calculated. The two test are illustrated in Figures 9 and 10 and the values of the relative errors are 4.345% and respectively 3.61%. Similarly, the same signals were applied to the other inputs and the same operations were performed. For this cases we have the situations shown in Figures 11 and 12 and the values of the relative nonlinearity errors are 4.438% and respectively 5.12%. In order to find the total harmonic distorsions (THD) of this circuit, a 1MHz sinusoidal signal was applied to one of the inputs and the other input was made 1V. The frequency spectrum of the output voltage is shown in Figure 13 and the THD due to the second and third harmonics was calculated to be less then 2%. Using the standard 40 pin analog frame provided by MOSIS, a 5x5 convolution unit and a single multiplier cell were combined into a single microchip. The 5x5 convolution unit will be tested for a Laplacian of Gaussian (LOG) kernel. The kernel values were generated using the LOG formula:

$$LOG(x,y) = \frac{2\sigma^2 - x^2 - y^2}{2\sigma^2} e^{\frac{-x^2-y^2}{2\sigma^2}} \qquad (9)$$

The values for the kernel were calculated for σ=1 and are shown next:

-0.0549	-0.1231	-0.1353	-0.1231	-0.0549
-0.1231	0.0000	0.3033	0.0000	-0.1231
-0.1353	0.3033	1.0000	0.3033	-0.1353
-0.1231	0.0000	0.3033	0.0000	-0.1231
-0.0549	-0.1231	-0.1353	-0.1231	-0.0549

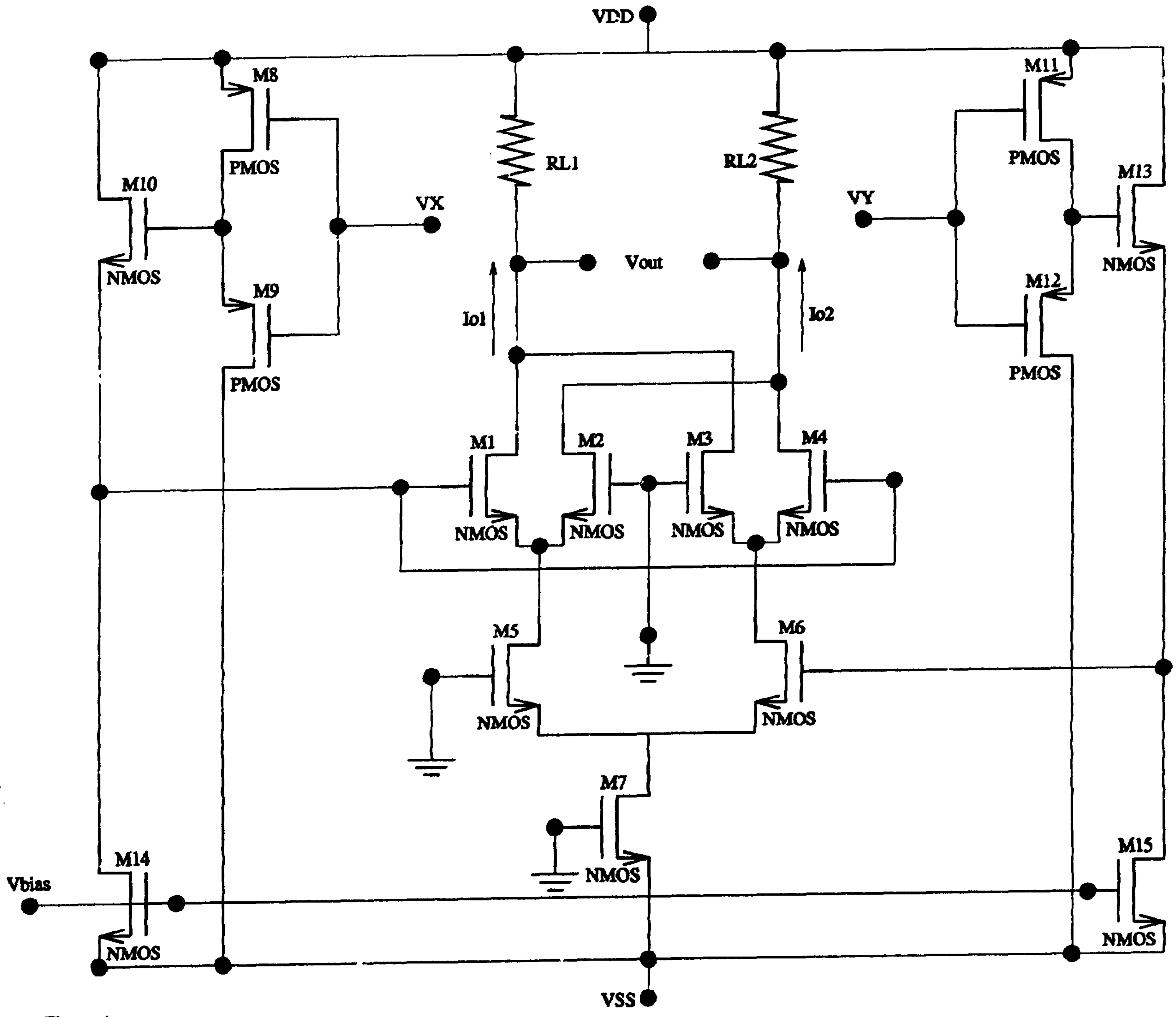
A simple voltage divider circuit (Figure 14) to implement this LOG kernel was incorporated in the microchip and connected to the Y_{ij} inputs, were $1 \le i \le 5$ and $1 \le j \le 5$. The SPICE file of the voltage divider can be found in Appendix B. All the X_{ij} inputs were connected to the analog input pads. The load resistors were not put in each individual cell. Instead, the corresponding ($Io1_i$ and $Io2_i$, $1 \le i \le 25$) differential current outputs were connected together to provide with the sum of them. Thus, the convolution unit performs also the summation of the products of the 25 multiplier cells using KCL. The result of the convolution can be accessed on Io1 and Io2 pins and is a differential current . which can be easily converted to a differential voltage using three resistors. This result is, therefore, a voltage which is proportional to $\sum_{i=1}^{25}\sum_{j=1}^{25} X_{ij}Y_{ij}$.

REFERENCES

[1] S.-C. Quin and R. L. Geiger, "A ±5V CMOS analog multiplier", *IEEE J. Solid-State Circuits*, vol. SC-22, No. 6, pp. 1143-1146, Dec. 1987.

[2] H.-J. Song and C.-K. Kim, "An MOS four-quadrant analog multiplier using simple two-input squaring circuits with source followers", *IEEE J. Solid-State Circuits*, vol. 25, pp. 841-848, No. 3, June 1990.

[3] J. S. Pena-Finol and J. A. Conelly, "A MOS four-quadrant analog multiplier using the quarter-square technique", *IEEE J. Solid-State Circuits*, vol. SC-22, No. 6, pp. 1064-1073, Dec. 1987.


VDD
M8
PMOS
M10
NMOS
M9
PMOS
VX
RL1
RL2
Vout
Io1
Io2
M11
PMOS
M13
NMOS
M12
PMOS
VY
M1
NMOS
M2
NMOS
M3
NMOS
M4
NMOS
M5
NMOS
M6
NMOS
M7
NMOS
M14
NMOS
Vbias
M15
NMOS
VSS
Figure 1

TRANSFER CHARACTERISTIC OF THE MULTIPLIER FOR A VX DC SWEEP

Date/Time run: 10/12/94 00:35:57

Temperature: 27.0

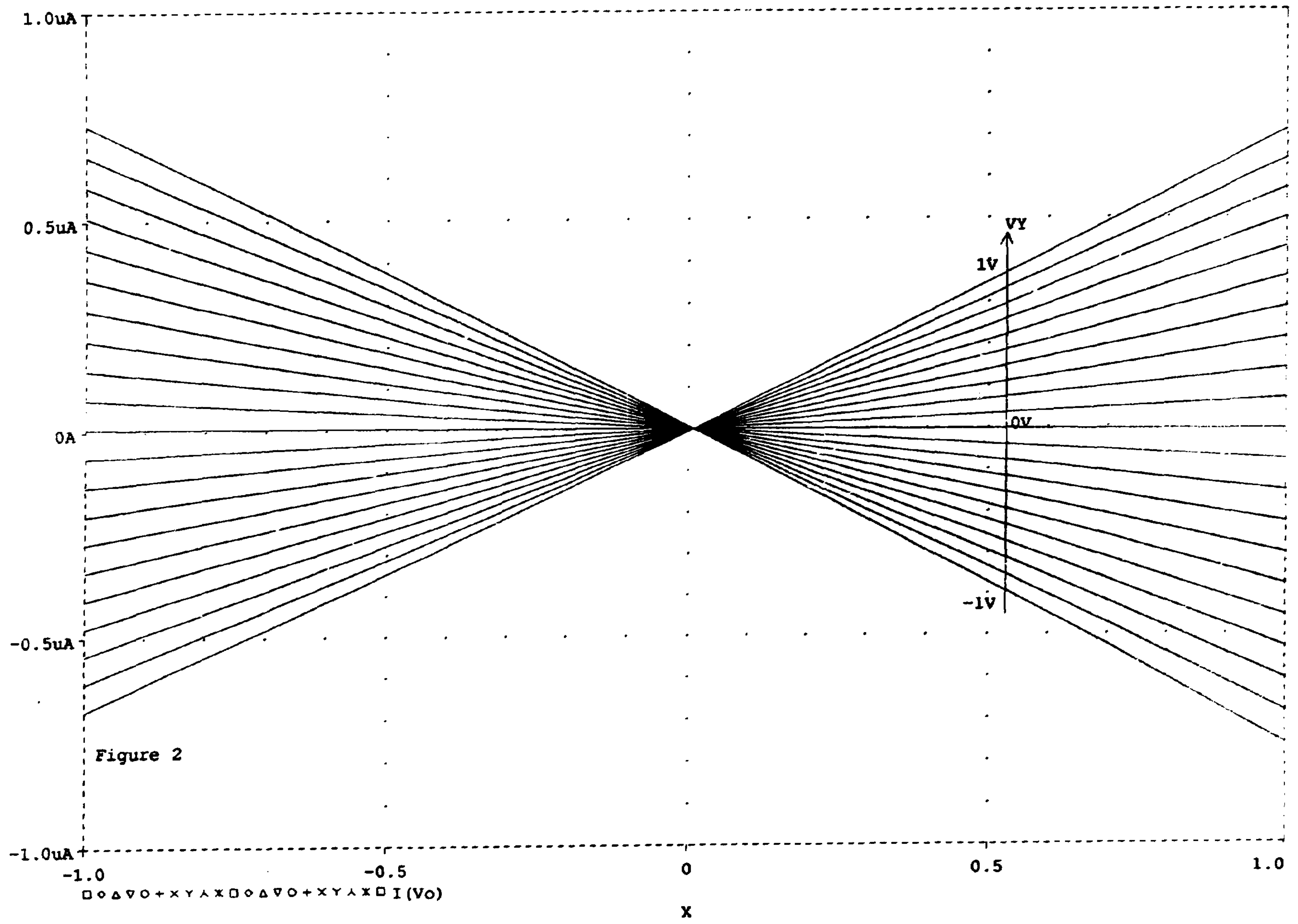


Figure 2

TRANSFER CHARARCTERISTIC OF THE MULTIPLIER FOR A VY DC SWEEP

Date/Time run: 10/12/94 00:29:46 Temperature: 27.0

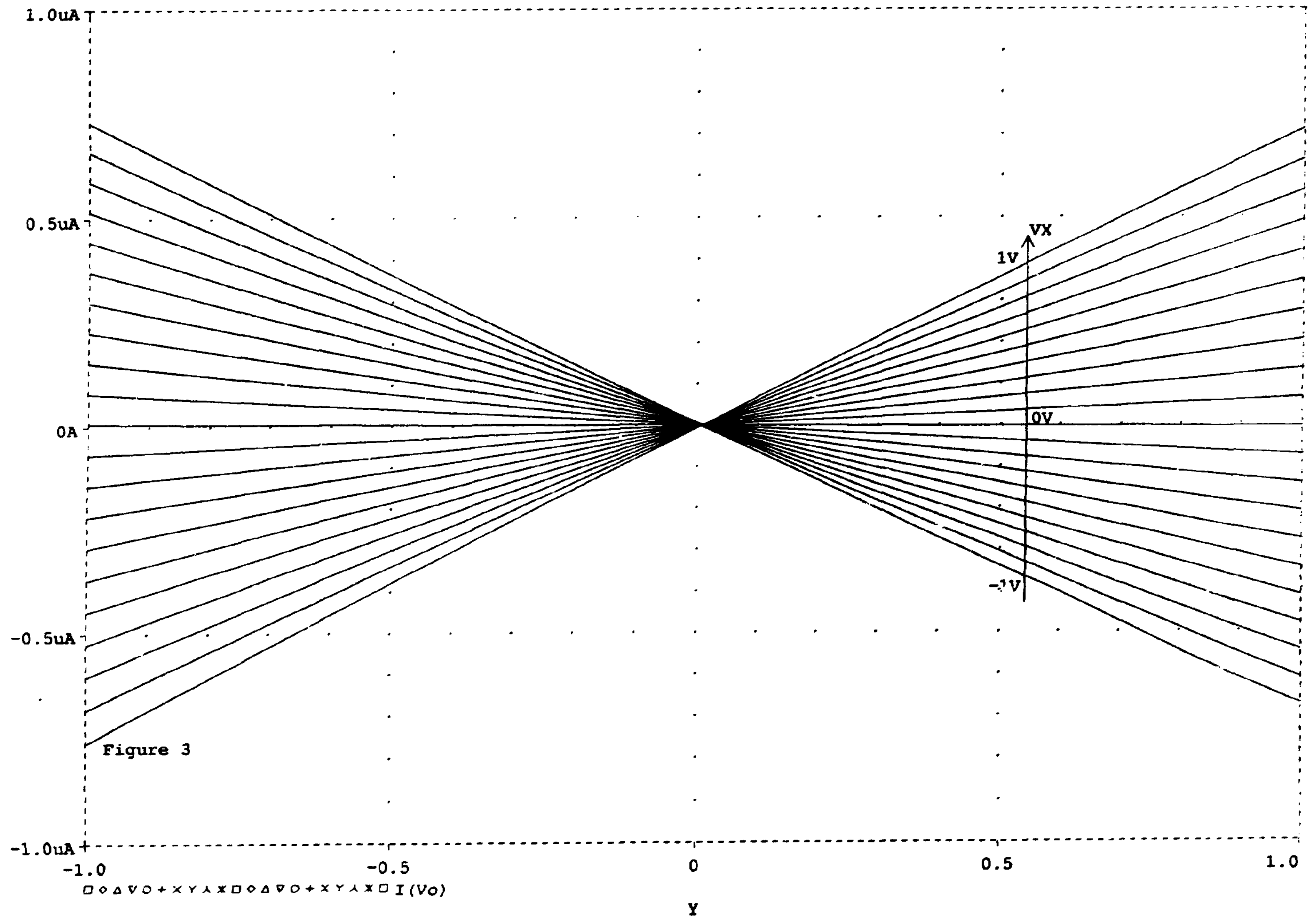


Figure 3

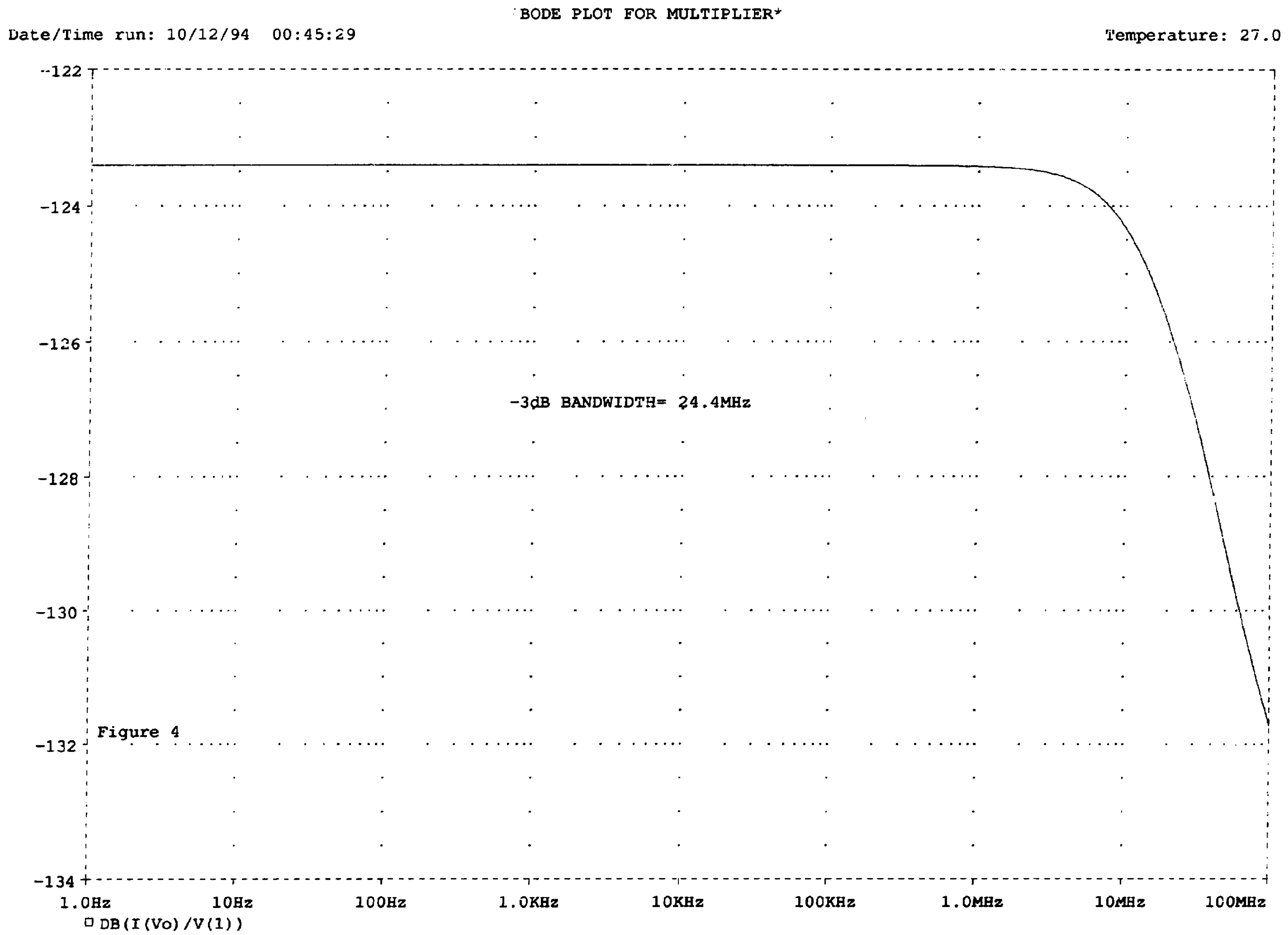

BODE PLOT FOR MULTIPLIER*
Date/Time run: 10/12/94 00:45:29
Temperature: 27.0
-122
-124
-126
-128
-130
-132
-134
-3dB BANDWIDTH= 24.4MHz
1.0Hz
10Hz
100Hz
1.0KHz
10KHz
100KHz
1.0MHz
10MHz
100MHz
□ DB(I(Vo)/V(1))
Frequency

Figure 4

EXTRACTED MULTIPLIER TRANSFER CHARACTERISTIC FOR A VY DC SWEEP

Date/Time run: 10/12/94 00:57:02 Temperature: 27.0

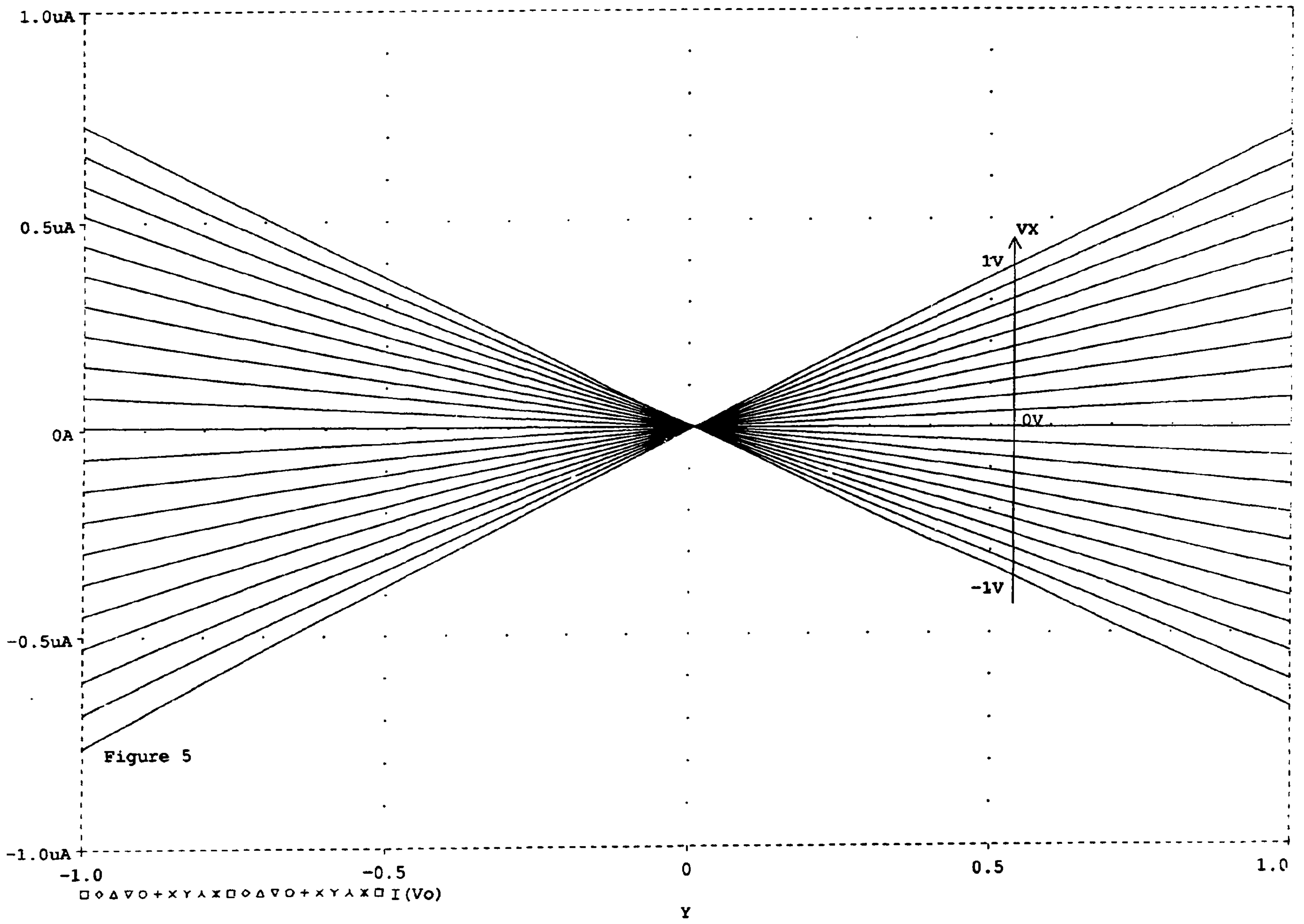


Figure 5

EXTRACTED MULTIPLIER TRANSFER CHARACTERISTIC FOR A VX DC SWEEP

Date/Time run: 10/12/94 01:00:25

Temperature: 27.0

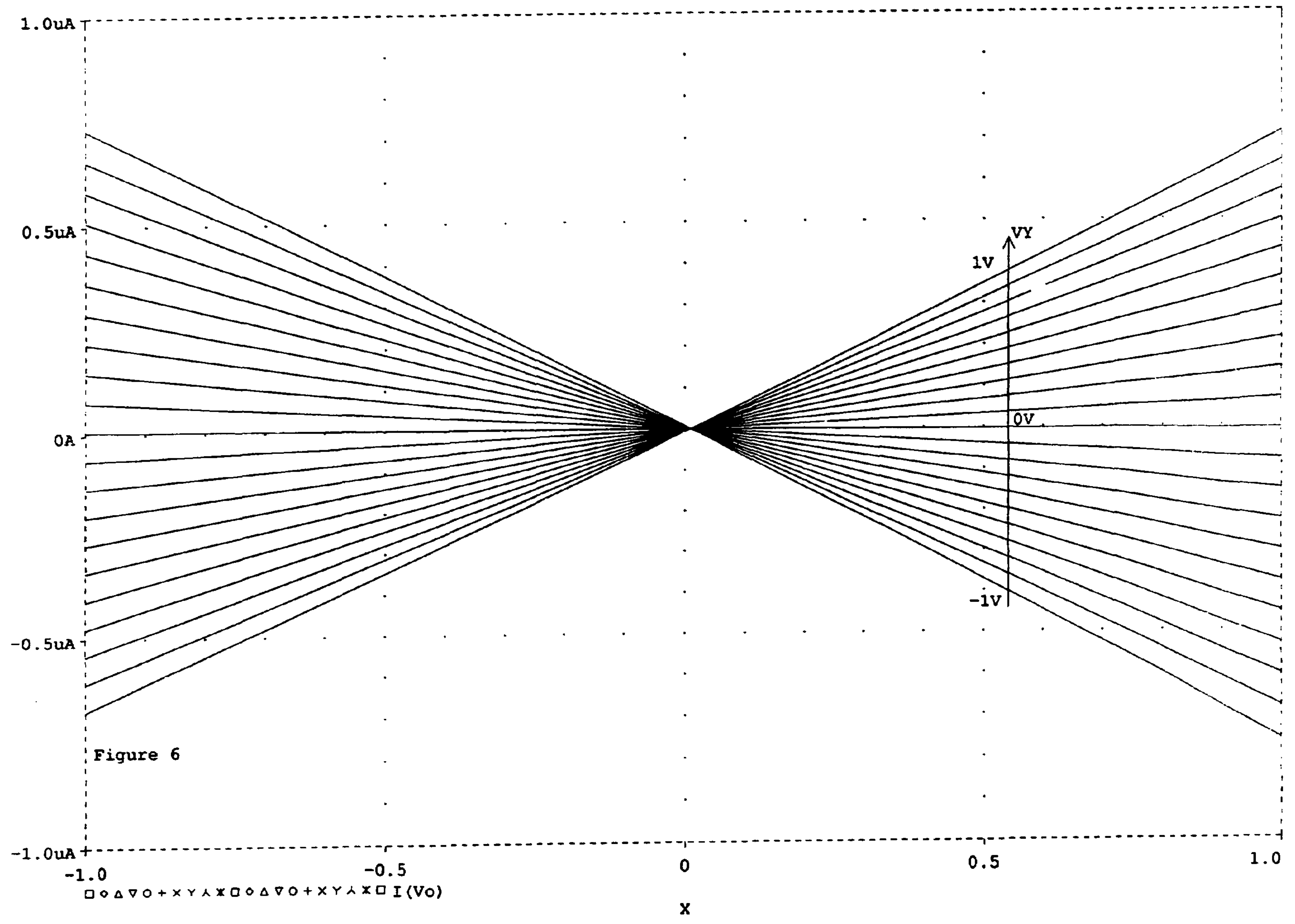


Figure 6

BODE PLOT OF THE EXTRACTED MULTIPLIER

Date/Time run: 10/12/94 01:03:53

Temperature: 27.0

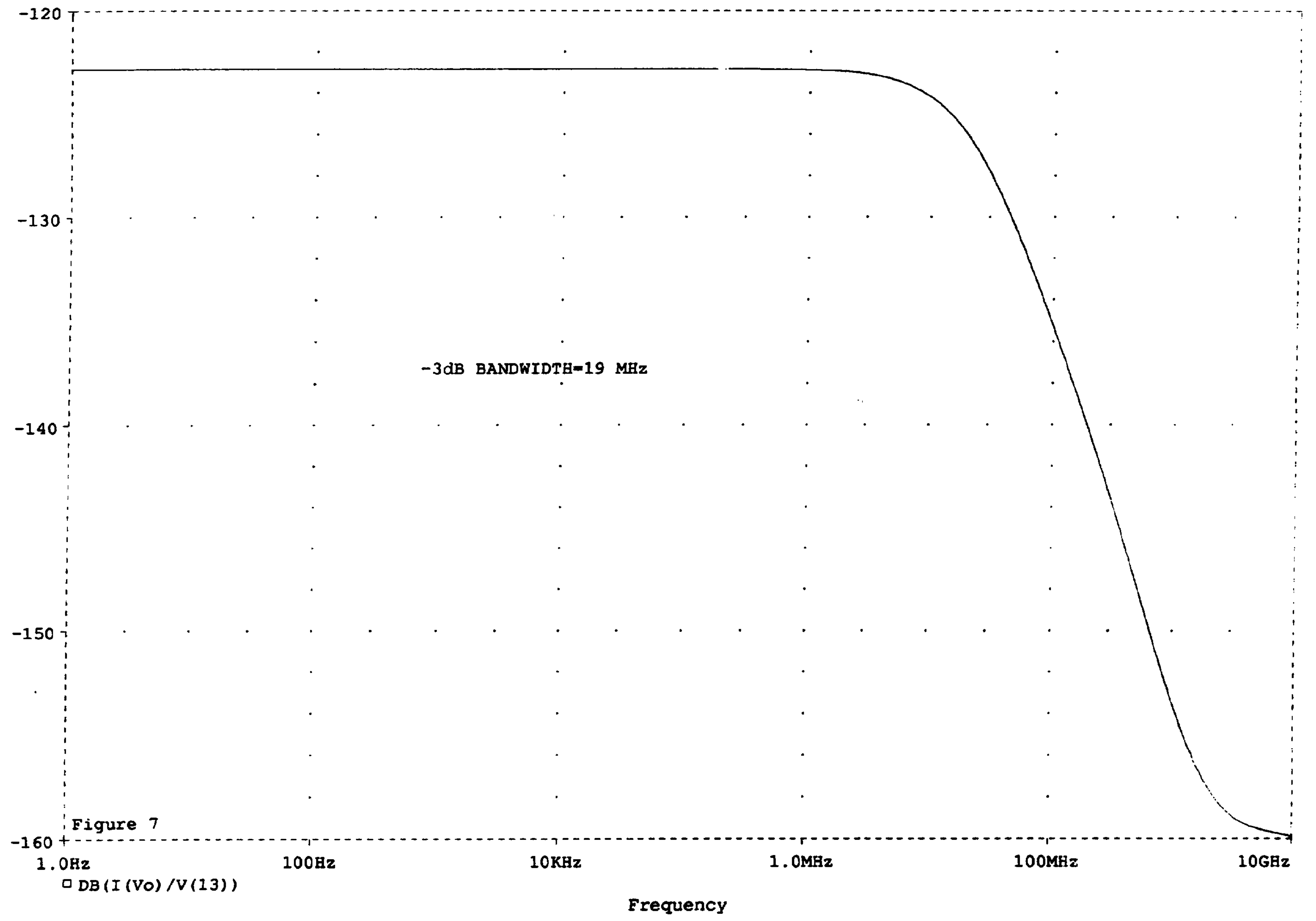
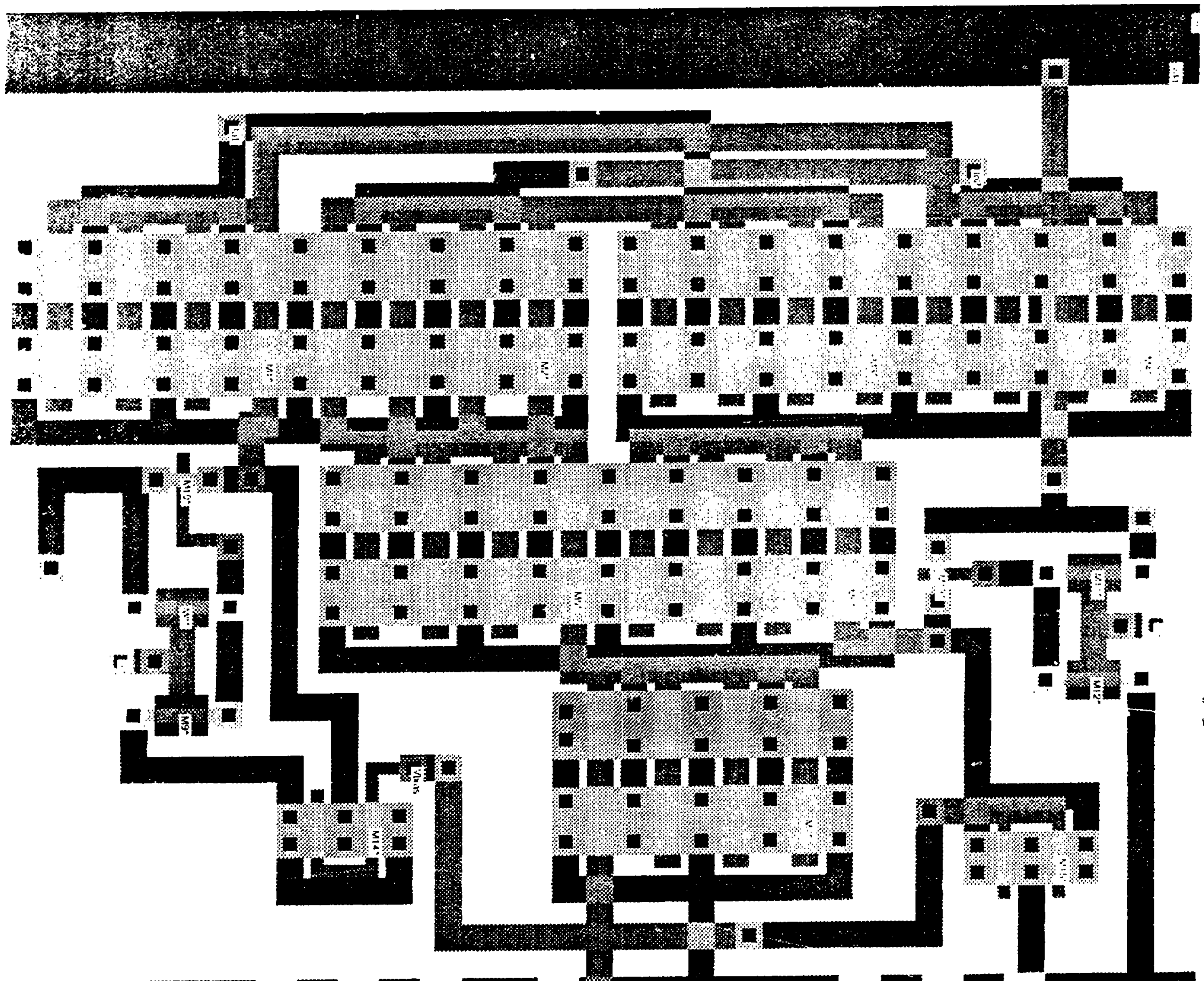


Figure 7


M9*
M10*
M11*
M12*
M14*
Vbias

[illegible]

*NONLINEARITY ERROR OF ANALOG MULTIPLIER

Date/Time run: 10/15/94 11:06:38

Temperature: 27.0

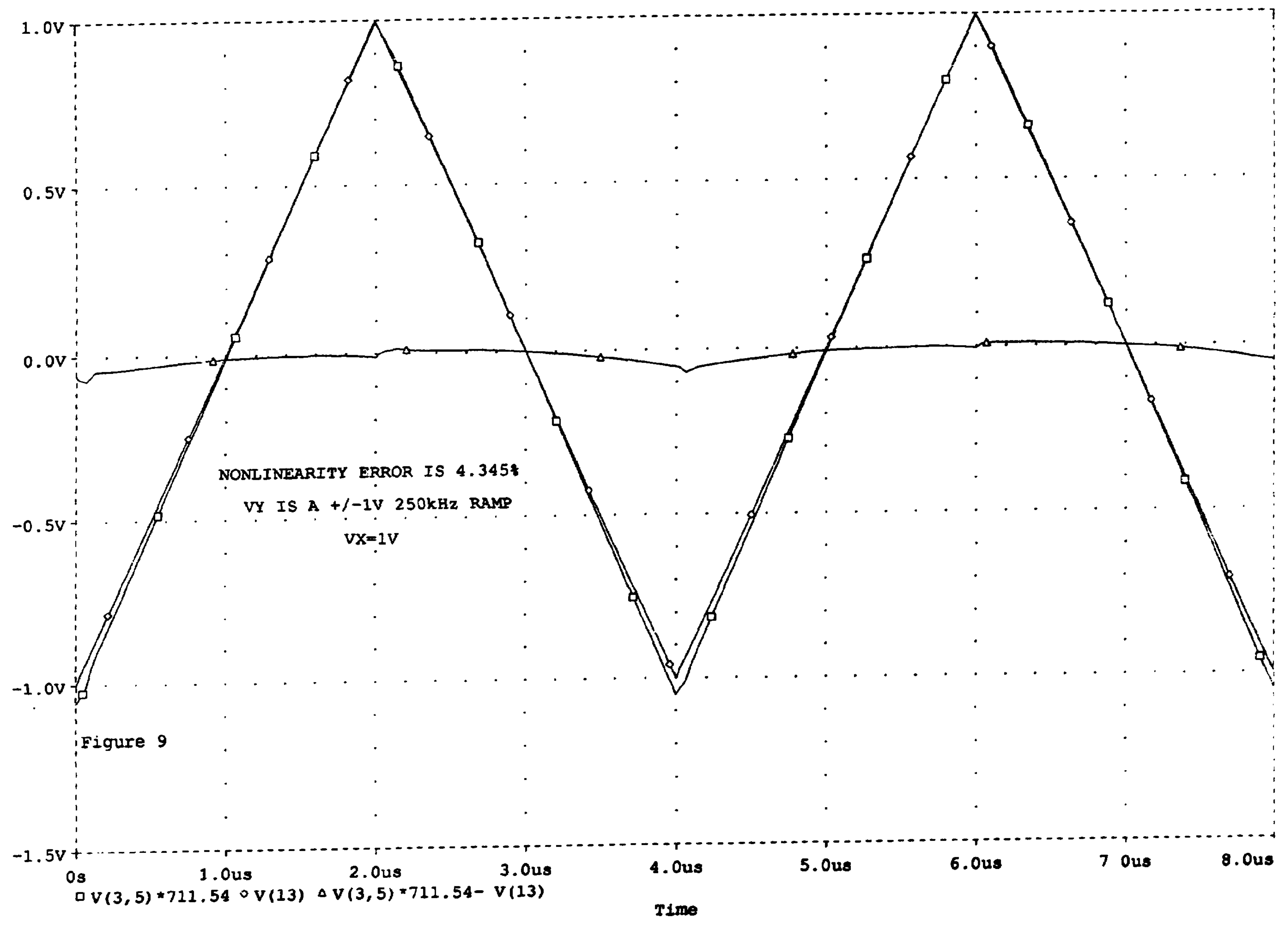


Figure 9

NONLINEARITY ERRORS OF ANALOG MULTIPLIER

Date/Time run: 10/15/94 11:06:38

Temperature: 27.0

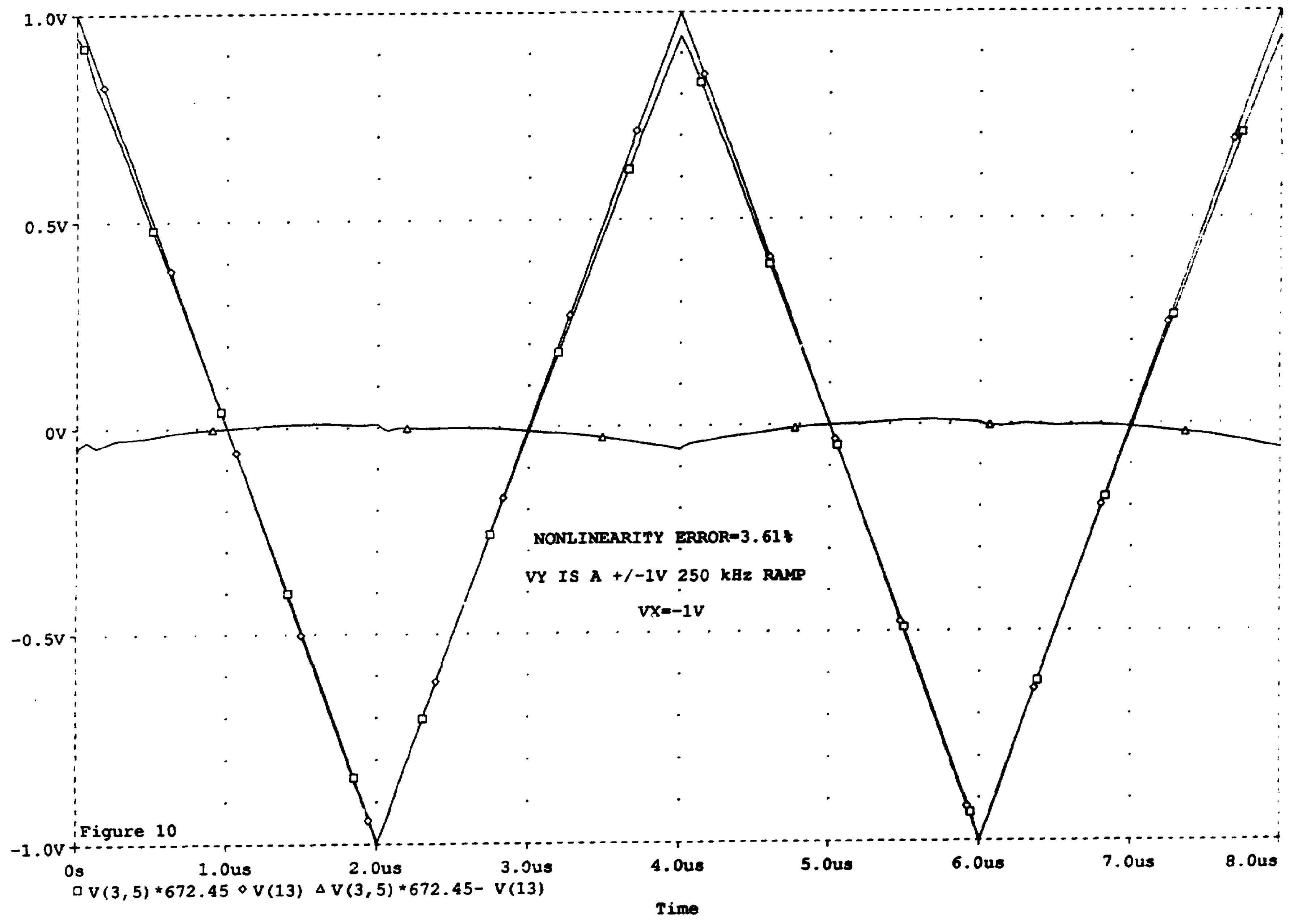
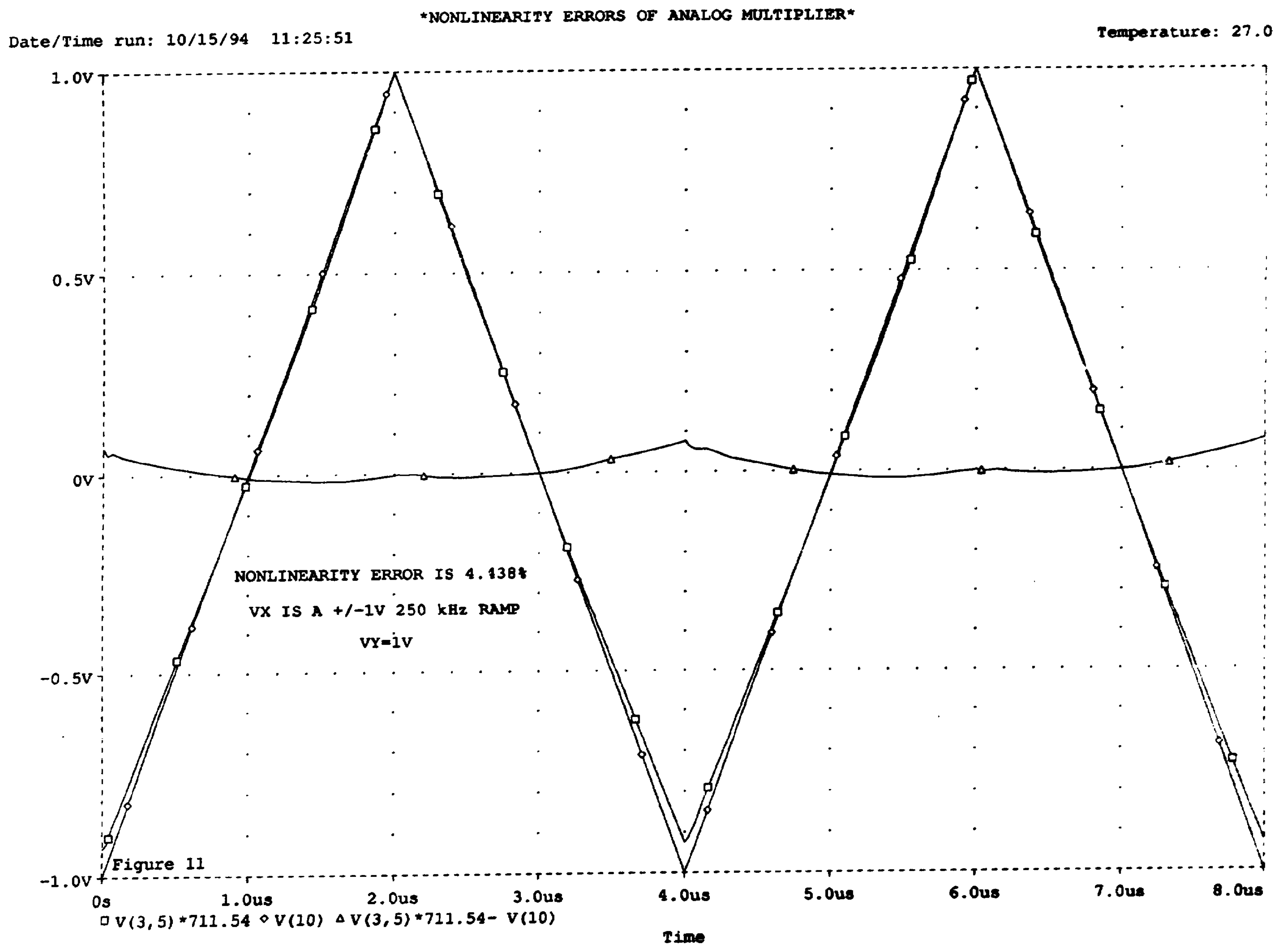


Figure 10


NONLINEARITY ERRORS OF ANALOG MULTIPLIER
Date/Time run: 10/15/94 11:25:51
Temperature: 27.0
1.0V
0.5V
0V
-0.5V
-1.0V
0s
1.0us
2.0us
3.0us
4.0us
5.0us
6.0us
7.0us
8.0us
Time
NONLINEARITY ERROR IS 4.138%
VX IS A +/-1V 250 kHz RAMP
VY=1V
□ V(3,5)*711.54 ◇ V(10) △ V(3,5)*711.54- V(10)
Figure 11

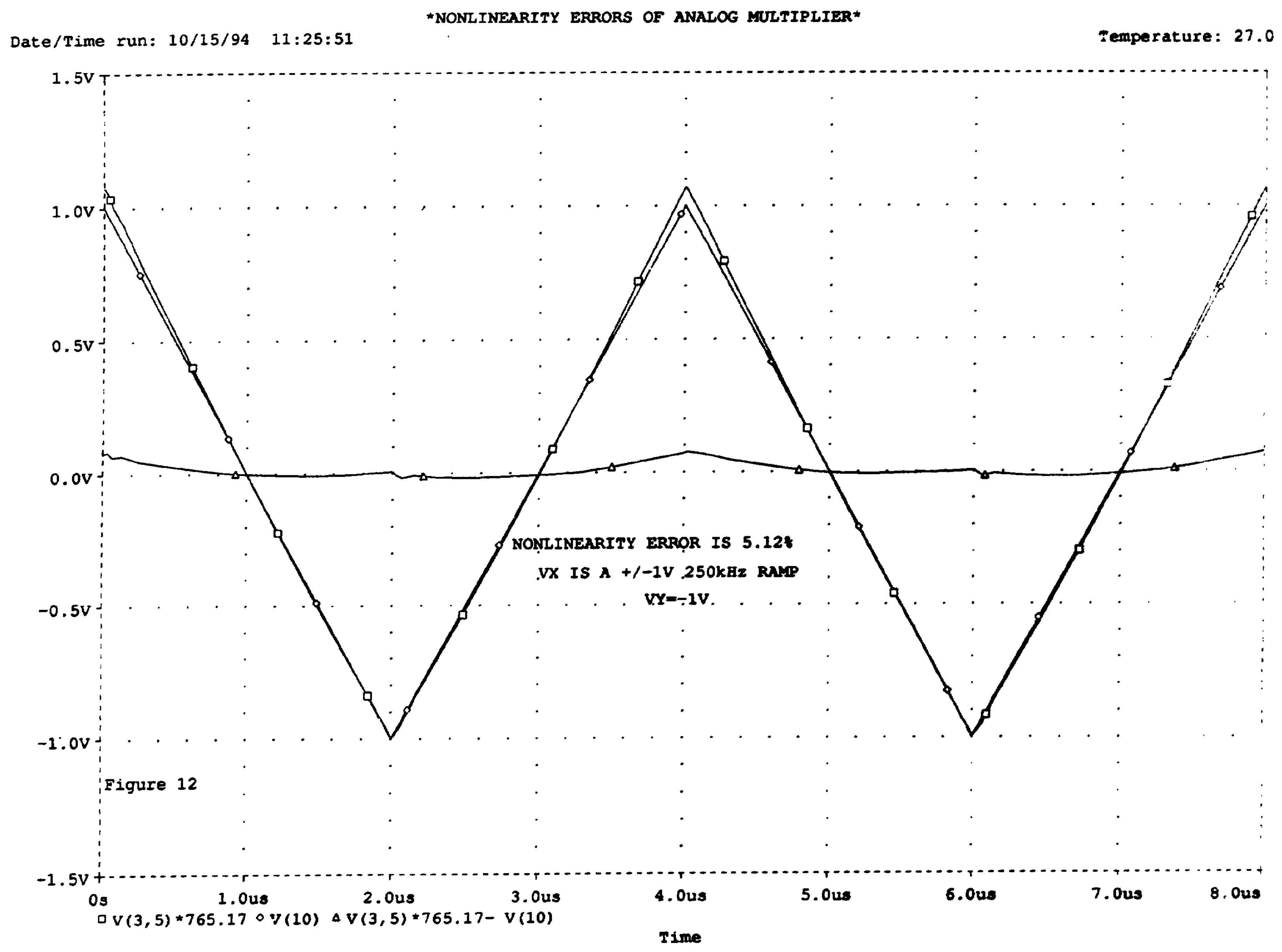

NONLINEARITY ERRORS OF ANALOG MULTIPLIER
Date/Time run: 10/15/94 11:25:51
Temperature: 27.0
1.5V
1.0V
0.5V
0.0V
-0.5V
-1.0V
-1.5V
0s
1.0us
2.0us
3.0us
4.0us
5.0us
6.0us
7.0us
8.0us
NONLINEARITY ERROR IS 5.12%
VX IS A +/-1V 250kHz RAMP
VY=-1V
□ V(3,5)*765.17 ◇ V(10) △ V(3,5)*765.17- V(10)
Time

Figure 12

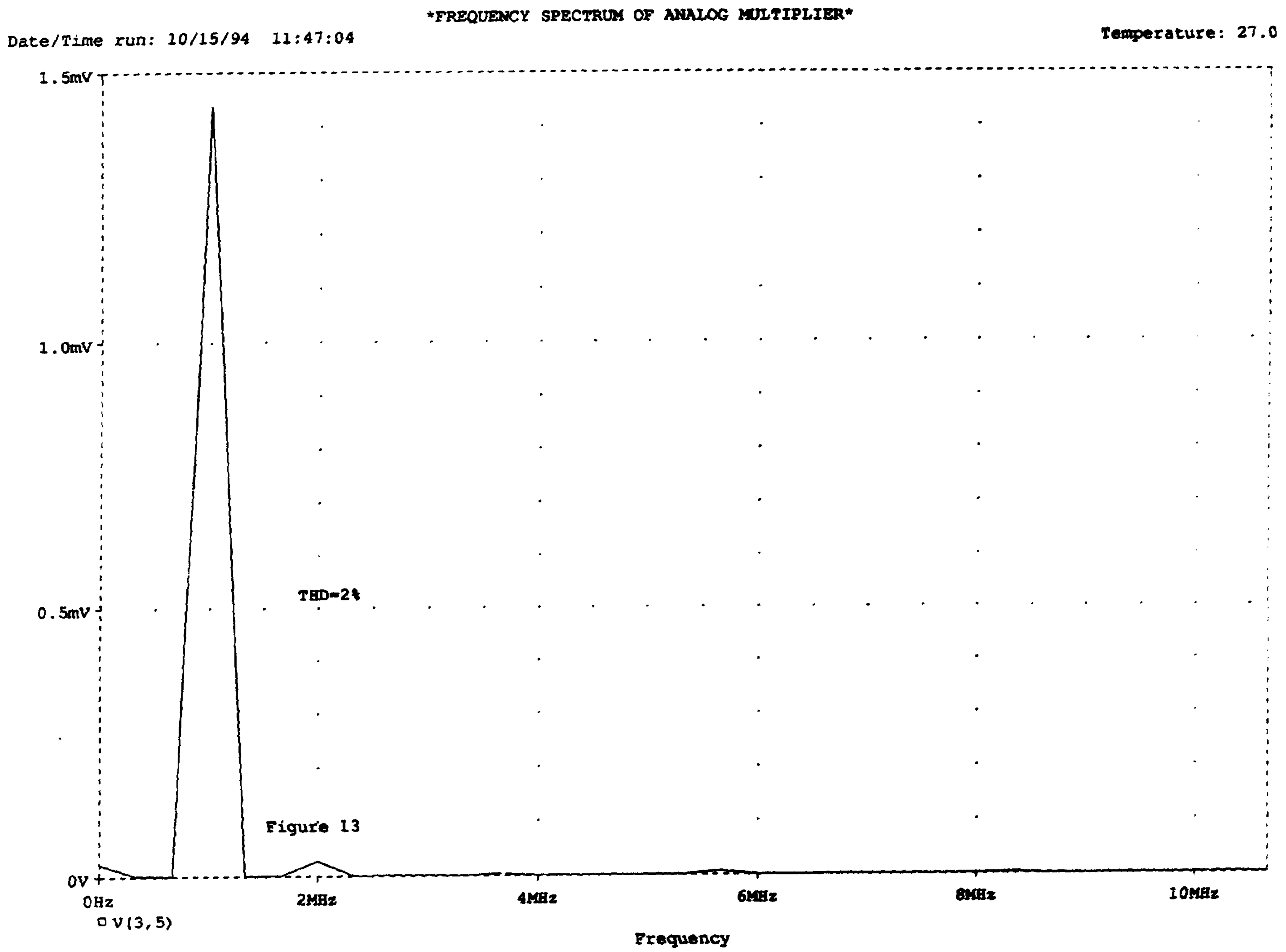

FREQUENCY SPECTRUM OF ANALOG MULTIPLIER
Date/Time run: 10/15/94 11:47:04
Temperature: 27.0
1.5mV
1.0mV
0.5mV
0V
THD=2%
0Hz
2MHz
4MHz
6MHz
8MHz
10MHz
□ V(3,5)
Frequency

Figure 13

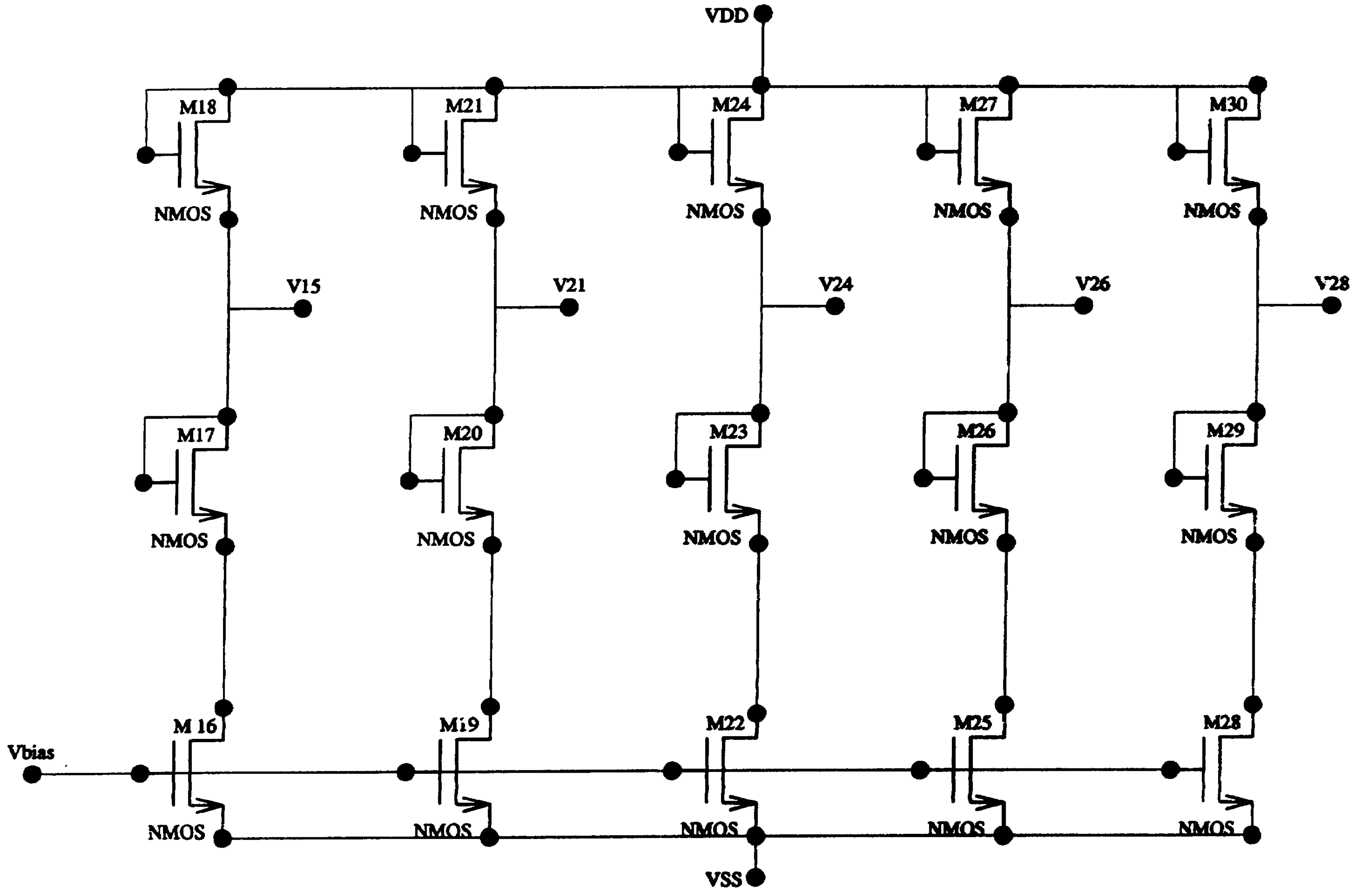
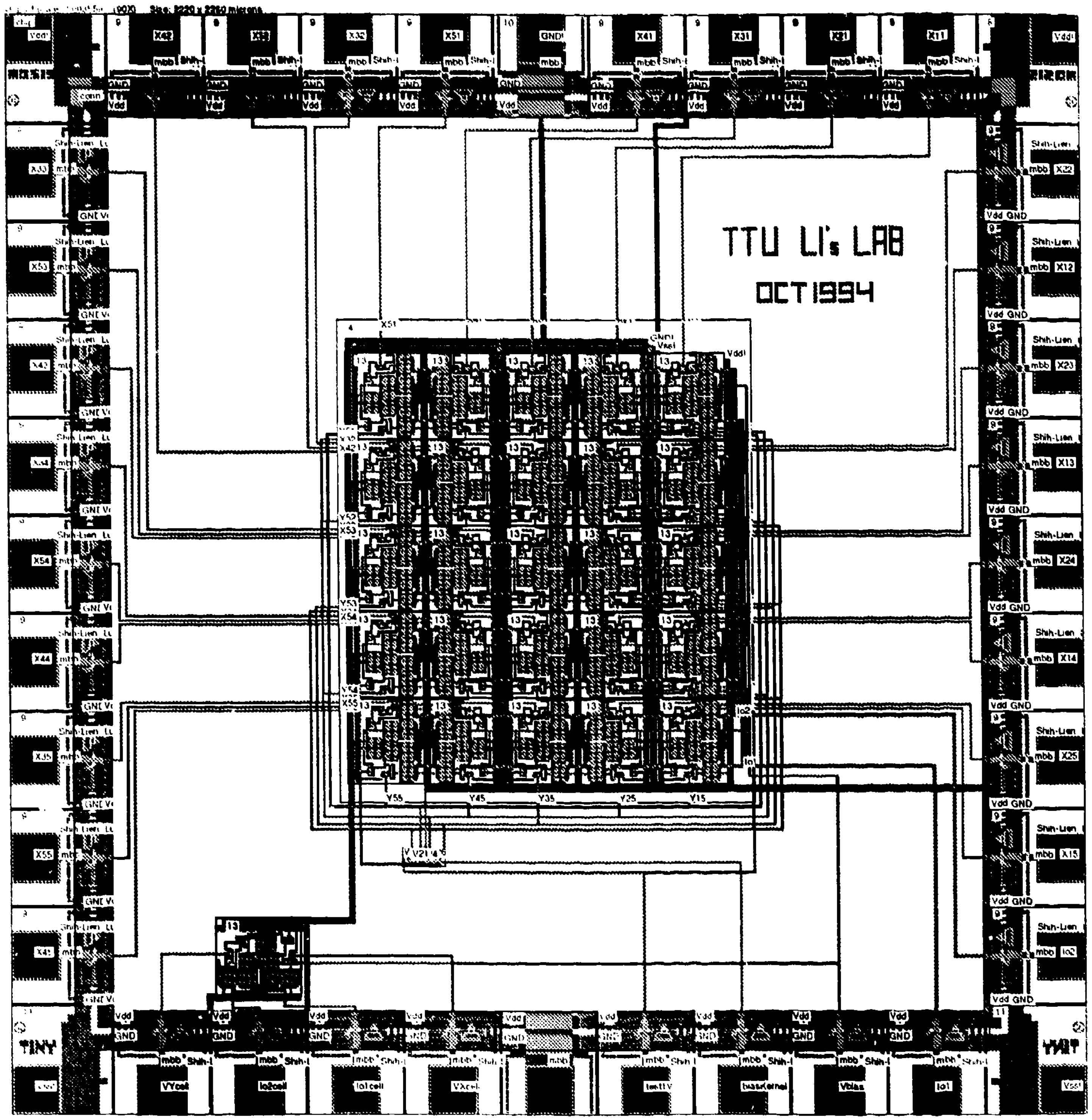

VDD
M18
M21
M24
M27
M30
NMOS
V15
V21
V24
V26
V28
M17
M20
M23
M26
M29
M 16
Mi9
M22
M25
M28
Vbias
VSS

Figure 14


(90X) Size: 2220 x 2260 microns
TTU LI's LAB
OCT 1994
X42
X52
X32
X51
GND!
X41
X31
X21
X11
Vdd!
X33
X22
X53
X12
X43
X23
X34
X13
X54
X24
X44
X14
X35
X25
X55
X15
X45
Io2
VYcell
Io2cell
Io1cell
VXcell
testV
biasKernel
Vbias
Io1
Vss!
Y55
Y45
Y35
Y25
Y15
MOSIS
TINY

APPENDIX A

```
*ANALOG MULTIPLIER WITH ASYMMETRIC INPUTS

.OPTION NOECHO NOMOD

*DEFINITION OF MODELS

*N43B  SPICE LEVEL 2 PARAMETERS

.MODEL N NMOS LEVEL=2 PHI=0.600000 TOX=4.2600E-08 XJ=0.200000U TPG=1
+ VTO=0.8109 DELTA=6.7500E+00 LD=9.2070E-08 KP=4.8109E-05
+ UO=593.5 UEXP=1.4990E-01 UCRIT=8.1140E+04 RSH=2.4540E+01
+ GAMMA=0.4512 NSUB=4.0300E+15 NFS=1.98E+11 VMAX=6.2580E+04
+ LAMBDA=2.6210E-02 CGDO=1.1195E-10 CGSO=1.1195E-10
+ CGBO=4.7024E-10 CJ=1.0922E-04 MJ=0.8608 CJSW=2.4755E-10
+ MJSW=0.035834 PB=0.800000
* Weff = Wdrawn - Delta_W
* The suggested Delta_W is -7.8340E-07
.MODEL P PMOS LEVEL=2 PHI=0.600000 TOX=4.2600E-08 XJ=0.200000U TPG=-1
+ VTO=-1.0600 DELTA=9.0900E+00 LD=1.7890E-07 KP=2.2267E-05
+ UO=274.7 UEXP=3.4430E-01 UCRIT=6.9200E+04 RSH=5.9420E+01
+ GAMMA=0.3746 NSUB=2.7770E+15 NFS=3.23E+11 VMAX=9.9990E+05
+ LAMBDA=5.3990E-02 CGDO=2.1752E-10 CGSO=2.1752E-10
+ CGBO=3.9953E-10 CJ=3.1585E-04 MJ=0.5914 CJSW=3.2974E-10
+ MJSW=0.315516 PB=0.700000
* Weff = Wdrawn - Delta_W
* The suggested Delta_W is -3.4000E-07
****************************************************************************
*---------------------------
*GILBERT CELL
*---------------------------
M1 3 2 1 4 N W=80U L=4U
M2 5 17 1 4 N W=80U L=4U
M3 3 17 6 4 N W=80U L=4U
M4 5 2 6 4 N W=80U L=4U
M5 1 18 7 4 N W=80U L=4U
M6 6 8 7 4 N W=80U L=4U
M7 7 19 4 4 N W=80U L=4U

*---------------------------
*ATTENUATOR 1
*---------------------------
M8 11 10 9 9 P W=2U L=3U
M9 4 10 11 9 P W=2U L=8U
M10 9 11 2 4 N W=2U L=3U
M14 2 16 4 4 N W=16U L=2U

*---------------------------
*ATTENUATOR 2
*---------------------------
M11 12 13 9 9 P W=2U L=3U
M12 4 13 12 9 P W=2U L=8U
M13 9 12 8 4 N W=2U L=3U
M15 8 16 4 4 N W=16U L=2U

RL1 3 9 1K
RL2 5 9 1K
RL 3 5 10MEG

********************************

Vdd 9 0 5V
Vss 4 0 -5V
Vbias 16 0 {VB}

V17 17 0 0V
V18 18 0 0V
```

```
V19 19 0 0V

*VX 10 0 {X}
VX 10 0 SIN(0 1 1MEG)
*VX 10 0 AC 1V
*VX 10 0 PWL(0US {-X} 2US {X} 4US {-X} 6US {X} 8US {-X} 10US {X})
*VX 10 0 PWL(0NS 0V 200NS 0V 200.001NS 1V 600NS 1V 600.001NS 0V 1US 0V)
VY 13 0 {Y}
*VY 13 0 SIN(0 1 1MEG)
*VY 13 0 AC 1V
*VY 13 0 PWL(0NS {-Y} 2US {Y} 4US {-Y} 6US {Y} 8US {-Y} 10US {Y})

.PARAM X=1V
.PARAM Y=1V
.PARAM VB=-3.915

*.STEP PARAM Y -1 1 .1
*.DC PARAM Y -1 1 .1
.TRAN/OP .01PS 1US
*.AC DEC 100 1 1E8
*.WATCH DC
.OP
*.PROBE V(14) V(18)
.END
```

APPENDIX B

```
*voltage dividers for kernel coefficients*

.OPTION NOECHO NOMOD

*DEFINITION OF MODELS

*N43B  SPICE LEVEL 2 PARAMETERS

.MODEL N NMOS LEVEL=2 PHI=0.600000 TOX=4.2600E-08 XJ=0.200000U TPG=1
+ VTO=0.8109 DELTA=6.7500E+00 LD=9.2070E-08 KP=4.8109E-05
+ UO=593.5 UEXP=1.4990E-01 UCRIT=8.1140E+04 RSH=2.4540E+01
+ GAMMA=0.4512 NSUB=4.0300E+15 NFS=1.98E+11 VMAX=6.2580E+04
+ LAMBDA=2.6210E-02 CGDO=1.1195E-10 CGSO=1.1195E-10
+ CGBO=4.7024E-10 CJ=1.0922E-04 MJ=0.8608 CJSW=2.4755E-10
+ MJSW=0.035834 PB=0.800000
* Weff = Wdrawn - Delta_W
* The suggested Delta_W is -7.8340E-07
.MODEL P PMOS LEVEL=2 PHI=0.600000 TOX=4.2600E-08 XJ=0.200000U TPG=-1
+ VTO=-1.0600 DELTA=9.0900E+00 LD=1.7890E-07 KP=2.2267E-05
+ UO=274.7 UEXP=3.4430E-01 UCRIT=6.9200E+04 RSH=5.9420E+01
+ GAMMA=0.3746 NSUB=2.7770E+15 NFS=3.23E+11 VMAX=9.9990E+05
+ LAMBDA=5.3990E-02 CGDO=2.1752E-10 CGSO=2.1752E-10
+ CGBO=3.9953E-10 CJ=3.1585E-04 MJ=0.5914 CJSW=3.2974E-10
+ MJSW=0.315516 PB=0.700000
* Weff = Wdrawn - Delta_W
* The suggested Delta_W is -3.4000E-07
*********************************************************************

*-----------------------------
*KERNEL
*-----------------------------

*for 1V

M16 14 22 4 4 N W=10U L=2U
M17 15 15 14 4 N W=2U L=6U
M18 9 9 15 4 N W=3U L=2U

*for 0.3033V

M19 20 22 4 4 N W=8U L=2U
M20 21 21 20 4 N W=6U L=2U
M21 9 9 21 4 N W=2U L=2U

*for -0.055V

M22 23 22 4 4 N W=8U L=2U
M23 24 24 23 4 N W=7U L=2U
M24 9 9 24 4 N W=2U L=3U

*for -0.123V

M25 25 22 4 4 N W=7U L=2U
M26 26 26 25 4 N W=2U L=6U
M27 9 9 26 4 N W=2U L=4U

*for -0.1353V

M28 27 22 4 4 N W=7U L=2U
M29 28 28 27 4 N W=2U L=4U
M30 9 9 28 4 N W=2U L=4U

*******************************************************

VDD 9 0 5V
VSS 4 0 -5V
```

```
Vbias 22 0 {VB}

.PARAM VB=-3.5V

*.STEP PARAM VB -3.6 -3.4 .01
*.DC PARAM VB -4 -3 .1
.TRAN 1PS 2NS
*.PROBE
.OP
.END
```

USER GUIDE: MOTION DETECTION ALGORITHM BASED ON GABOR FUNCTIONS

USER GUIDE: MOTION DETECTION ALGORITHM BASED ON GABOR FUNCTIONS

Professor Hua Harry Li
Computer Science Department
College of Engineering
Texas Tech University
Lubbock, Texas, USA

CONTENTS

5 SOFTWARE LISTS

PREFACE

This software manual describes the algorithms that have been developed for the demonstration project on motion detection and motion compensation using biologically inspired Gabor transforms. The objectives of the project are to (1) develop an algorithm suitable for real time analog VLSI (Very Large Scale Integrated Electronics Circuits) implementation and (2) fabricate the design through MOSIS. The algorithms developed and tested in this package will be used as bench marks to objectively evaluate the performance of the VLSI chips, and the prototype board. This package is a part of the result of the research project supported by ONR.

This package includes three categories of programs: (1) The programs that are used to generate test pattern images. (2) The programs that are developed to compute optical flows by using Gabor functions, or Gabor Transforms. And (3) the programs that are used to generate sequence of images by using virtual reality and three-dimensional computer graphics techniques. During the process of preparing this package, we have extensively tested each program. Most of the programs in this package were written in ANSI C to ensure the portability across different hardware platforms. For the programs to generate 3D virtual reality studio environment, we have used the program language developed by Watkins [1], which is a C-type language. Although most of the programs were developed for SUN SPARC Station, they can also be complied for 486 personal computer with some minor modifications. The use of these programs for ONR is free, but we, the authors of these programs and manual, make no warranty of any kind, expressed or implied, with regard to the programs or the documentation. Therefore, the authors shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs. All brand names, trademarks, are the property of their respective holders.

Several research assistants at Texas Tech University are responsible for developing, implementing, and testing of the programs. The present version of

[1]* C.D. Watkins, S.B. Coy, and M. Finlay, *Photorealism and Ray Tracing in C*, M&T Books, New York, NY 10011, 1992.

the manual was proof read and tested at Li's Laboratory, Computer Science Department, College of Engineering, Texas Tech University by the principal investigator of the project. Inquiry and questions regarding the source code, the algorithms should be directed to the principal investigator.

Principal Investigator: Hua Harry Li, Ph.D.
Computer Science Department
Collge of Engineering
Lubbock, Texas 79409
Phone: (806) 742-3513
Fax: (806) 742-3519
E-mail: hua@cs.ttu.edu

1 INTRODUCTION

ABSTRACT

A brief summary of the programs and their functionalities included in this manual are given in this chapter. Namely, three major categories of programs are described. These categories are: (1) the programs for generating test pattern images as bench marks, (2) the programs for computing optical flow based on Gabor functions, or Gabor transforms, and (3) the programs based on virtual reality technique for generating 3D camera models and creating user-defined and user-controlled image patterns.

1 BRIEF OVERVIEW

This package includes three categories of programs:

1. The programs that are used to generate test pattern images. The images are characterized with user-defined orientation and spatial frequence.
2. The programs that are developed to compute optical flows by using Gabor functions, or Gabor Transforms. In this category, there are utility programs for solving a set of linear algebraic system by using least square estimation (LSE) techniques, and programs for user selectable, user modifiable Gabor kernels.
3. The programs that are used to generate sequence of images by using virtual reality and three-dimensional computer graphics techniques. Users are given the control of selecting the optical characteristics of a virtual camera

to generate well-controlled camera motion, as ego motion. This then is used to precisely characterize the amount of self motion in comparison to object motion.

In this documentation, we use the term "two-dimensional motion" to describe the image pattern generated directly by shifting image plane, which does not involve the selection of camera model, the calculation of perspective projection. While the term "three-dimensional motion" referres to as the image patterns generated by using *virtual reality* technique. Namely, the technique defines camera characteristics and creats realistic looking perspective projection images. The two-dimensional motion patterns are majorly used to calibrate the orientation selectivity and spatial frequency selectivity of the Gabor kernels.

2 PREPARING PROGRAM FOR EXECUTION

The compiled and linked executables of the programs are provided on a floppy disk (MSDOS readable format) with this manual. These executables are prepared under the following conditions:

1. The operating system: SunOS 4.1.2.
2. The cc compiler.
3. The hardware platform: SUN SPARC 1 workstation.

If you have the complied and linked executables, in order to run these programs all you have to do is to type the program name and hit return. If you only have the source code and you need to compile and link them, then follow the instructions given below:

```
cc -o output-filename inputfile.c -lm
```

which will creat the executable.

2 GENERATING 2D TEST PATTERN IMAGES

ABSTRACT

Two different types of test pattern images, 2D test patterns, and 3D test patterns, have been utilized for testing the motion detection algorithm as explained in Sec'ion 1.2. The 2D test patterns are grey level sine, cosine, or strip bar images. These test patterns are used to calibrate, test and demonstrate the orientation and spatial frequency selectivity of Gabor kernels.

1 CREATING COSINE WAVE IMAGE PATTERNS

DECRIPTION: creat a 2-D cosine wave image.

USAGE: CosImg(int row, int column, float omega, float orient, float speed, float time)

> **int row:** row of image in pixels.
> **int column :** column of image in pixels.
> **float omega:** spatial frequency of image in cycles per pixel.
> **float orient:** orientation of image in degree.
> **float speed:** the variable to define the phase shift.
> **float time:** the variable for defining the phase shift.

REMARKS:

CosImg generates a 2-D cosine wave image with user selected resolution (row-by-column) and user defined spatial frequency and orientation of the cosine pattern.

MATHEMATICAL FORMULATION:

$$I(x, y) = cos(2 * pi * omega * (x * cos(theta) + y * sin(theta) - speed * time)).$$

where variable omega defines a spatial frequency in terms of cycles per pixel, variable theta defines an orientation of the test pattern. The variable speed defines the phase shift of the image pattern which can be interpreted as a starting time instance when the test pattern image was created. By choosing different phase shift, user can creat a sequence of images arranged in the order of smallest phase shift to the biggest phase shift to represent a sequence of moving cosine patterns. The variable time, together with the variable speed creats the desired phase shift.

EXAMPLE:

```
#include<stdlib>
#include<stdio.h>

int main()
{

char *filename="output.dat";
int row=31, column=31;
float spatial_frequency=.1, orientation=30;
float speed = 1, time = 0.4;

CosImg(row, column, spatial_frequency, orientation, speed, time);
return 0;
}
```

By changing the different phase shift value, the sine wave function can be created using the same program.

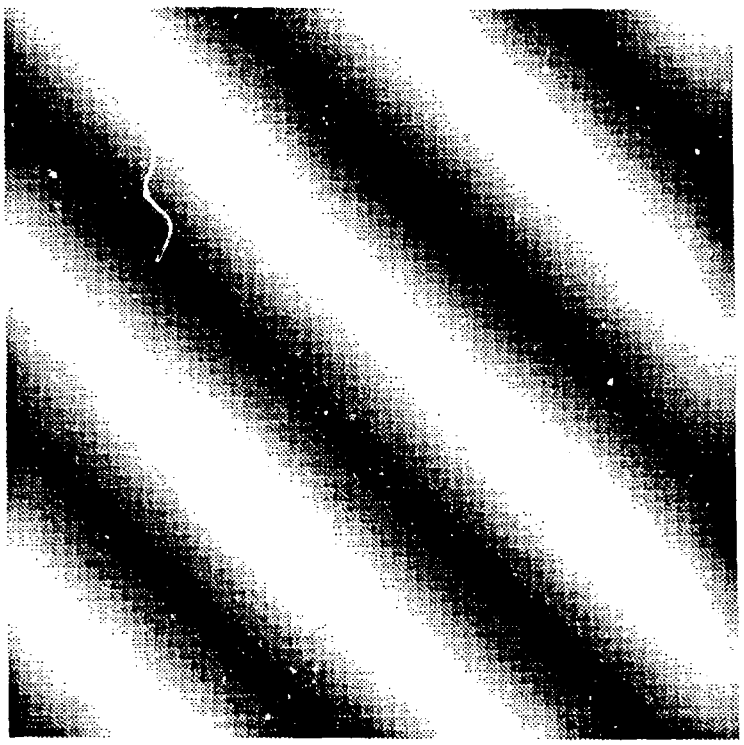


Figure 1: A cosine wave image (64 by 64 resolution) with omega equal to 0.05 and theta equal to 45 degree.

2 CREATING BAR STRIPE IMAGES

DESCRIPTION: generate a 2-D bar stripe image.

USAGE: BarImg(int row, int column, float omega, float orient, char* filename)

introw: row of image in pixels.
intcolumn : column of image in pixels.
floatomega: spatial frequency of image in cycles per pixel.
floatorient: orientation of image in degree.
char*filename: the name of output file.

REMARKS:

BarImg generates a 2-D bar stripe image with user-selected resolution (row by column) and user-selected spatial frequency and orientation.

MATHMATICAL FORMULATION:

$$I(x,y) = \begin{cases} 255 & \text{if } \cos 2\pi(x\omega_x + y\omega_y) > 0.0 \\ 0 & \text{otherwise} \end{cases}$$

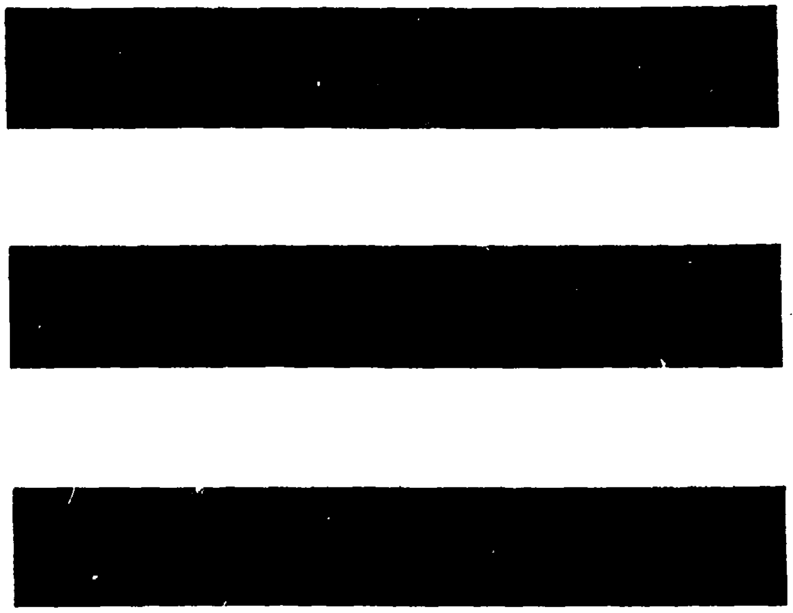


Figure 2. A bar strip image with omega equal to 0.05 and theta equal to 0 degree.

EXAMPLE:

```
#include<stdlib>
#include<stdio.h>

int main()
{

char *filename="output.dat";
int row=31, column=31;
float spatial_frequency=.1, orientation=30;

BarImg(row, column, spatial_frequency, orientation, filename);
return 0;
}
```

3

MOTION DETECTION BASED ON GABOR FUNCTIONS

ABSTRACT

This chapter gives the programs that implement motion detection algorithm based on Gabor functions. Gabor functions are very special kind of functions which mimics orientation selective, spatial frequency selective characteristics of human early vision system. Included here are the functions for creating Gabor kernels, the utility functions for computing convolution of real and complex Gabor kernels, the functions for solving a set of linear algebraic system based on least square estimation (LES), and the function for computing motion vectors, as well as the utility functions for ploting optical flow pattern.

1 CREATING GABOR KERNELS

DECRIPTION: Creat 2-D Gabor kernels and the derivatives (with respect to x and with respect to y) of Gabor kernels.

USAGE:
void CreateGabor(omega, theta, type, size, tempt, tempx, tempy)

> **float** omega: modulated spatial frequency (ω_0) (cycles per pixel).
> **float** theta: orientation of the kernel (degree).
> **int** type: kernel type: 1 for real kernel, 0 for imaginary kernel.
> **int** size: the desired kernel size.
> **float**** tempt: pointer of a block to Gabor kernel.

float** tempx: pointer of a block to the derivative (with respect to x) of the Gabor kernel.
float** tempy: pointer of a block to the derivative (with respect to y) of the Gabor kernel.

REMARKS:

CreateGabor() creats Gabor kernel and its derivative kernels (the derivative with respect to x or y.) The modulated spatial frequency, orientation, and size of the kernels are determined by the user selected omega, theta, and size. The result of the created kernel is stored in three blocks dynamically allocated in the memory.

MATHEMATICAL FORMULATION:

$$G_r(x,y) = e^{(-\frac{x^2+y^2}{\sigma^2})} cos 2\pi(\omega_x x + \omega_y y),$$
$$G_i(x,y) = e^{(-\frac{x^2+y^2}{\sigma^2})} sin 2\pi(\omega_x x + \omega_y y).$$

the derivatives of Gabor real and imaginary parts are

$$\frac{\partial G_r}{\partial x} = -2\pi e^{-\frac{x^2+y^2}{\sigma^2}}\left(\frac{x}{\sigma^2}\cos 2\pi(\omega_x x + \omega_y y) + \omega_x \sin 2\pi(\omega_x x + \omega_y y)\right),$$
$$\frac{\partial G_r}{\partial y} = -2\pi e^{-\frac{x^2+y^2}{\sigma^2}}\left(\frac{y}{\sigma^2}\cos 2\pi(\omega_x x + \omega_y y) + \omega_y \sin 2\pi(\omega_x x + \omega_y y)\right),$$
$$\frac{\partial G_i}{\partial x} = -2\pi e^{-\frac{x^2+y^2}{\sigma^2}}\left(-\frac{x}{\sigma^2}\sin 2\pi(\omega_x x + \omega_y y) + \omega_x \cos 2\pi(\omega_x x + \omega_y y)\right),$$
$$\frac{\partial G_i}{\partial y} = -2\pi e^{-\frac{x^2+y^2}{\sigma^2}}\left(-\frac{y}{\sigma^2}\sin 2\pi(\omega_x x + \omega_y y) + \omega_y \cos 2\pi(\omega_x x + \omega_y y)\right).$$

where ω_x and ω_y are modulated spatial frequency components in sptial frequency domain,

EXAMPLE:

```
#include<stdlib>
#include<stdio.h>
#include<math.h>
```

```
int main()
{
static float **kern_t, **kern_x, **kern_y;
float spatial_freq = .1, orientation = 0;
int filter_type = 0;
int row, col, kernsize, index;
FILE *outt, *outx, *outy;

  /* calculate kernel size and index */
  kernsize = (int)ceil(1.0/(spatial_freq));
  if(fabs((float)kernsize/2.0-ceil((float)kernsize/2.0))<=0.01)
  kernsize++;
  index=(int)(((float)kernsize-1.0)/2.0);

  /* allocate memory for kernels */
  kern_t =(float **) malloc(kernsize*sizeof(float*));
  kern_x =(float **) malloc(kernsize*sizeof(float*));
  kern_y =(float **) malloc(kernsize*sizeof(float*));
  for (i=0; i< kernsize; i++ ) {
   kern_t[i] = (float *)malloc(kernsize*sizeof(float)+1);
   kern_x[i] = (float *)malloc(kernsize*sizeof(float)+1);
   kern_y[i] = (float *)malloc(kernsize*sizeof(float)+1);
   }

  /* generate Gabor type kernels */
   CreateGabor(spatial_freq, orientation, filter_type, index, kern_t, kern_x, kern_y);

  if((outt = fopen("gabor_t.dat", "w")) == NULL) exit(1);
  if((outx = fopen("gabor_x.dat", "w")) == NULL) exit(1);
  if((outy = fopen("gabor_y.dat", "w")) == NULL) exit(1);
    for(i=0; i< kernsize; i++) {
      for (j=0; j< kernsize; j++) {
          fprintf(outx, "%f\t", kern_t[i][j]);
          fprintf(outx, "%f\t", kern_t[i][j]);
          fprintf(outx, "%f\t", kern_t[i][j]);
         }
         fprintf(outx, "\n");
         fprintf(outx, "\n");
         fprintf(outx, "\n");
      }

    fclose(outt);
    fclose(outx);
    fclose(outy);
    return 0;
  }
```

2 COMPUTING CONVOLUTION OF GABOR KERNELS

2.1 Image Buffer Initialization

DECRIPTION: initialize a buffer for convolution.
USAGE: void BufferInit(index, kernsize, column, buffer, fp)

int index: index of Gabor type kernel.
int kernsize: Gabor kernel size
int column: column of the given image.
unsigned char** buffer: pointer of the block storing image.
FILE* fp: pointer of an image file.

REMARKS:

BufferInit() initializes a buffer for convolution. The use of a buffer accomplishes the convolution of a given image one row at a time. The size of the buffer can be determined by the product of the number of rows of a given kernel and the number of columns of the given image. The top half space of the buffer is initialized to be filled with the first row of the given image for the consideration of convolution with image boundary and another half of the buffer was filled with the image data. The image data should be 8-bit per pixel in binary format. The buffer is a static unsigned char double pointer type.

EXAMPLE:

```
#include<stdlib.h>
#include<stdio.h>
```

```
#include<malloc.h>

int main()
{
static unsigned char **imagebuffer;
int image_row=50, image_col=50;
int  kernel_index=7, kernel_size = 15;
int i, j;
FILE *in;

  /* allocate memory for image buffer */
  imagebuffer =(unsigned char **) malloc(image_row*sizeof(unsigned char*));
  for (i=0; i< image_col; i++ ) {
   imagebuffer[i] = (unsigned char *)malloc(image_col*sizeof(unsigned char));
   }

  /* open image file in READ_ONLY and BINARY mode */
  if((in = fopen("image.dat", "rb")) == NULL) exit(1);

  /* initialize image buffer */
   BufferInit(kernel_index, kernel_size, image_col, imagebuffer, in);

    for(i = 0; i < kernel_size; i++ )
       for (j =0; j < image_col; j ++ )
         printf("Image Data [%d][%d] = %d\n", i, j, imagebuffer[i][j]);

   return 0;
  }
```

2.2 Image Buffer Shifter

DECRIPTION: update the content of the image buffer by shifting up one row. This shift will result row 1 of the buffer being discarded, row 2 moved to row 1, row 3 moved to row 2, etc., and a new image row moved to the last row of the buffer. The shift is performed every time the convolution of one row is finished.

USAGE:
void BufferShifter(kern_index, kern_size, row_img, col_img, row_loop, buffer, fp)

int kern_index: index of Gabor kernel.
int kern_size: Gabor kernel size.
int row_img: row of image.
int col_img: column of image.
int row_loop: actual row number in convolution double loop.
unsigned char** buffer: pointer of a block storing image.
FILE* fp: pointer of the image file.

REMARKS:

BufferShifter() shifts data of each row in image buffer up to next row and updates data of last row in image buffer after each convolution operation. The row_loop is the actual row number in convolution double loop in order to consider the lower boundary in the convolution operation. The image format should be 8-bit per pixel in binary, and the image buffer could be static unsigned char double pointer type.

EXAMPLE:

```
#include<stdlib.h>
#include<stdio.h>
#include<malloc.h>

int main()
{
static unsigned char **imagebuffer;
int image_row=50, image_col=50;
int  kernel_index=7, kernel_size = 15;
int i, j;
FILE *in;

  /* allocate memory for image buffer */
  imagebuffer =(unsigned char **) malloc(image_row*sizeof(unsigned char*));
  for (i=0; i< image_col; i++ ) {
   imagebuffer[i] = (unsigned char *)malloc(image_col*sizeof(unsigned char));
   }

  /* open image file in READ_ONLY and BINARY mode */
  if((in = fopen("image.dat", "rb")) == NULL) exit(1);
```

```
 /* initialize image buffer */
  BufferInit(kernel_index, kernel_size, image_col, imagebuffer, in);
 /* display data in image buffer before shifting */
   for(i = 0; i < kernel_size; i++ )
      for (j =0; j < image_col; j ++ )
        printf("Image Data [%d][%d] = %d\n", i, j, imagebuffer[i][j]);
 /* shift 1 row of data in image buffer */
  BufferShifter(kernel_index, kernel_size, image_row, image_col, 10, imagebuffer, in);

/* display data in image buffer after shifting */
   for(i = 0; i < kernel_size; i++ )
      for (j =0; j < image_col; j ++ )
        printf("Image Data [%d][%d] = %d\n", i, j, imagebuffer[i][j]);
  free(imagebuffer);
  return 0;
 }
```

2.3 Image Convolution

DECRIPTION: perform 2D convolution.

USAGE:
float ConvolutionUnit(kern_index, col_image, col_loop, image_buf, kern_buf)

int kern_index: index of Gabor type kernel.
int col_image: column of image.
int col_loop: actual column number in convolution double loop.
unsigned char** image_buf: pointer of a block storing image.
unsigned char** kern_buf: pointer of a block storing kernel.

REMARKS:

ConvoiutionUnit() performs 2D convolution with image data from image buffer and kernel data from kernel buffer. The column boundary in image plane has been taken care of in this module, while the row boundary in image plane has been taken care of by BufferShifter().

The col_loop is the actual column number in convolution double loop. After convolution operation, the image buffer is updated by Buffer-Shifter() so that next convolution operation will be performed accordingly with the input of right set of image data. The image format is 8-bit per pixel in binary, and the image buffer is static unsigned char double pointer type.

EXAMPLE:

```
#include<stdlib.h>
#include<stdio.h>
#include<math.h>
#include<malloc.h>

int main()
{
static unsigned char **imagebuffer;
static float **kern_t, **kern_x, **kern_y;
float spatial_freq = .1, orientation = 0;
int filter_type = 0;
int row, col, kernsize, index;
int image_row=50, image_col=50;
float image;
FILE *in, *out;

  /* calculate kernel size and index */
  kernsize = (int)ceil(1.0/(spatial_freq));
  if(fabs((float)kernsize/2.0-ceil((float)kernsize/2.0))<=0.01)
  kernsize++;
  index=(int)(((float)kernsize-1.0)/2.0);


  /* allocate memory for kernels */
  kern_t =(float **) malloc(kernsize*sizeof(float*));
  kern_x =(float **) malloc(kernsize*sizeof(float*));
  kern_y =(float **) malloc(kernsize*sizeof(float*));
  for (i=0; i< kernsize; i++ ) {
   kern_t[i] = (float *)malloc(kernsize*sizeof(float)+1);
   kern_x[i] = (float *)malloc(kernsize*sizeof(float)+1);
   kern_y[i] = (float *)malloc(kernsize*sizeof(float)+1);
}

/* allocate memory for image buffer */
imagebuffer =(unsigned char **) malloc(kernsize*sizeof(unsigned char*));
for (i=0; i< image_col; i++ ) {
 imagebuffer[i] = (unsigned char *)malloc(image_col*sizeof(unsigned char));
 }

/* generate Gabor type kernels */
CreateGabor(spatial_freq, orientation, filter_type, index, kern_t, kern_x, kern_y);


/* open image file in READ_ONLY and BINARY mode */
if((in = fopen("image.dat", "rb")) == NULL) exit(1);
if((out = fopen("conv.dat", "w")) == NULL) exit(1);

/* initialize image buffer */
 BufferInit(kernel_index, kernel_size, image_col, imagebuffer, in);

/* do convolution */
 for( row = 0; row < image_row; row ++ ) {
  for ( col = 0; col < image_col; col ++ ) {
   image = ConvolutionUnit(index, image_col, col, imagebuffer, kern_t);
   fprintf(out, "%f\t", image);
   /* shift data in image buffer */
   BufferShifter(index, kernsize, image_row, image_col, row, imagebuffer, in);
  }
   fprintf(out, "\n");
}

 free(kern_t);
 free(kern_x);
 free(kern_y);
 free(imagebuffer);
 return 0;
}
```

3 MOTION DETECTION ON 2D TEST PATTERN IMAGES

This section describes a motion detection with one orientation and spatial frequency tuned Gabor function. Unlike the algorithm to be described in the next section, this motion detection is majorly used as testing purpose for the know sine or cosine wave image patterns, which can be tuned to only one orientation and spatial frequency.

DESCRIPTION: motion detection on a cosin wave moving images.

USAGE: GaborCos

input:
(1) Modulated Spatial frequency (ω_0) of the Kernel (Cycles per pixel)
(2) Length of the kernel in sigma value (length = 1/(sigma*omega))
(3) Kernel Orientation
(4) Thresold for Motion Detection (to eliminate extreme high value of motion (u,v) from the homogeneous image region where derivative features are poorly defined
(5) Kernel Phase (0: Gabor Real Part; 90: Gabor Imaginary Part)
(6) Derivative of Image File: file name of the derivative (finite difference) of the image frame 3 minus frame 1
(7) Image File: file name of the 2nd image frame

output:
(1) u.dat motion speed along X axis
(2) v.dat motion speed along Y axis
(3) t1.dat: gabor response to derivative of moving image (Ω_t)
(4) x1.dat: gradient gabor response to moving image (Ω_x)
(5) y1.dat: gradient gabor response to moving image (Ω_y)

The kernel size and image size are defined in #define macros.

REMARKS:

This program extracts an motion vector by Gradient Gabor Model. A set of three consecutive image frames is needed for extracting motion information. The first frame and the third frame are used to form image derivative by finite difference technique, then the difference is convolved with the Gabor kernel to produce Ω_t. The second frame is convolved with derivatives of Gabor (in x and in y direction) to produce Ω_x and Ω_y. The user has to first define the orientation and spatial frequency of the cosine test pattern. The selection of the length of the Gabor kernel has to be made with the consideration to the orientation and spatial frequency selectivity. As a general guideline, the user is advised to use the following formula to estimate the length value:

$$Length = \frac{1}{\Sigma}$$

where Σ is the sigma of the test pattern.

Another point for the user to realize is that thresholding is performed on the processed image data, Ω_x, Ω_y, and Ω_t, to deal with very small value at certain pixel location. With this thresholding the calculated optical flow vectors at these locations will be free of erroneous results due to the nature of the early vision computing (most of them are mathematically characterized as *ill-posed* problems).

MATHMATICAL FORMULATION:

$$\Omega_t = \Omega_x u + \Omega_y v$$

where Ω_t is the convolution of Gabor kernel with derivative of image, Ω_x is the convolution of x gradient Gabor kernel with moving image, Ω_y is the convolution of y gradient Gabor kernel with moving image.

EMAMPLE: CosConv

Modulated Spatial frequency In Kernel(Cycles/pixel): 0.1
Length: 4.0
Kernel Orientation(degree): 30.0
Thresold of Motion Detection: 100.0
Kernel Phase(degree): 90.0
Derivative of Image File: devi.img
Moving Image File: move.img

4 MOTION DETECTION ON 2D IMAGES WITH MULTIPLE KERNELS

For any realistic motion detection, different spatial frequency and orientation features should all be considered. This requires the use of more than one spatial frequency and orientation tuned Gabor kernels. This section describes how this multiple-kernel process can be accomplished.

DECRIPTION: Motion Detection on Moving Images.

```
#include "leastst.c"
```

USAGE: GaborMotion

input:
(1) Modulated spatial frequency (ω_0) of the kernel (cycles per pixel)
(2) Starting orientation of the kernel (degree)
(3) Rotating angles each time (degree)
(4) Number of rotations, or the number of different oriented Gabor kernels
(5) Thresold of motion detection (a recommendated value is greater than 50)
(6) Kernel phase (0: Gabor Real Part; 90: Gabor Imaginary Part)
(7) Derivative of Image File: file name of derivative (finite difference) of the average of image frame 3 minus frame 1 (8) Moving Image File: file name of image frame 2

output:
(1) u.dat motion speed along X axis
(2) v.dat motion speed along Y axis

The kernel size and image size are defined in #define macros.

REMARKS:

This program extracts an optical flow from 3 consecutive image frames by Gradient Gabor Model. When selecting different length of kernel, the consideration has to be given in the connection to the preprocessing of bandpass filtering or lowpass filtering operation. We suggest the use of Gaussian kernel for lowpass filtering or LoG (Laplacian of Gaussian) kernel for bandpass filtering before the use of Gabor kernels for motion detection. As one general guideline, the same sigma value of the Gabor kernel can be used for the Gaussian kernel, and the same sigma of the Gabor kernel can be used for the LoG kernel.

MATHEMARTCAL FORMULATION:

$$\vec{\Omega}_t + \vec{\Omega}_x u + \vec{\Omega}_y v = 0$$

where $\Omega_t = [\Omega_t^1, \Omega_t^2, ..., \Omega_t^k]^T$, $\Omega_x = [\Omega_x^1, \Omega_x^2, ..., \Omega_x^k]^T$, $\Omega_y = [\Omega_y^1, \Omega_y^2, ..., \Omega_y^k]^T$,
Ω_t^k is the convolution of Gabor kernel with derivative of image,
Ω_x^k is the convolution of x gradient Gabor kernel with moving image, and
Ω_y^k is the convolution of y gradient Gabor kernel with moving image.

EXAMPLE: GaborMotion

Modulated Spatial frequency In Kernel(Cycles/pixel): 0.1
Stating Orientation of Kernel(degree): 0.0
Rotating Angles Each Time(degree): 30.0
Number of Rotations: 4
Thresold of Motion Detection: 100.0
Kernel Phase(degree): 0.0
Derivative of Image File: devi.img
Moving Image File: move.img

5 LEAST SQUARE ESTIMATION

DESCRIPTION: Least-Square-Estimation

USAGE: solve2(int row, int column, float thresold)
or solve4(int row, int column, float thresold)

int row: the row of image;
int column: the column of the image;
float thresold: the thresold of motion detection.

REMARKS:

This program performs a least-square-estimation to minimize the error of optical flow computation. The technique will allow the better handling of the ill-posed problem, optical flow computation. As it is well known, many early vision problems are ill-posed in nature. To be able to deal with this, a least square estimation has been developed for optical flow computation with a set of 2 different orientation Gabor kernels or a set of 4 different orientation Gabor kernels respectively.

MATHMETICAL FORMULATION:

$$E(u,v) = \left|\vec{\Omega}_t + \vec{\Omega}_x u + \vec{\Omega}_y v\right|^2$$

EXAMPLE: solve2(60, 60, 200.0)

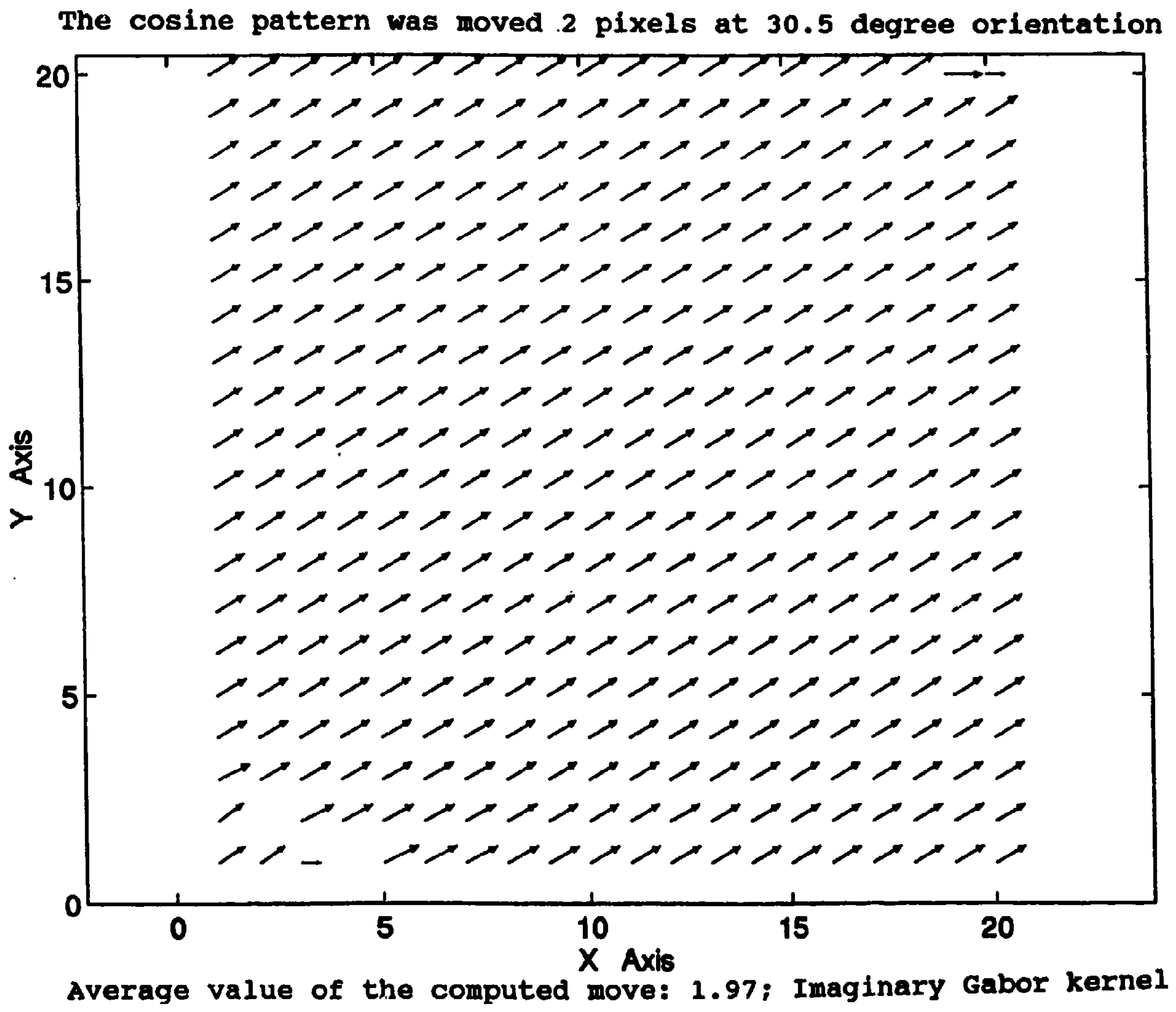


Figure 3. Motion detection result for cosine wave images, three frames were generated with orientation 30.5 degree and spatial frequency 0.025. The motion detection was performed by Gabor kernel with orientation and spatial frequency equal to 30.5 degree and 0.025 respectively.

4

THREE-DIMENSIONAL MOTION DETECTION

ABSTRACT

1 VIRTUAL REALITY TECHNIQUE FOR CREATING 3D TEST PATTERNS

To be able to test the motion detection algorithm realistically with better control of the testing condition, we have developed the technique based on the concept of virtual reality to creat virtual camera model. We have been using the tool developed by Watkins [1]. A C-type language was used to creat the user desired testing conditions which include the user selection of the following features:

- A world-coordinate system where every objects are defined and are referenced to, and a viewer-coordinate system which is observer-centered coordinate system, as well as the relative position between the two.
- A camera model with the choice of orientation (pointing direction) of the camera, the optical characteristics of the lens.
- Lighting condition with user control of the placement and the number of point light sources.
- 3D object construction using primitive building blocks, and the choice of different reflection models, such as diffused reflection, specular reflection, and ambient light.

[1] C.D. Watkins, S.B. Coy, and M. Finlay, *Photorealism and Ray Tracing in C*, M&T Books, New York, NY 10011, 1992.

Given in the floppy disk number 2, in directory MOVIE, one of the image sequence generated by this technique, ONR1.FLI, can be displayed as a movie on 486/50MHz machine with SuperVGA monitor. To display this sequence, first load the floppy disk into a floppy disk driver, then go to this drive and go to directory MOVIE, then type PLAY ONR1.FLI to execute the display of the movie. Please be aware of the software copyright issue, the program in this floppy disk should not be copyed, or used for any other purpose. To obtain a copy of this utility program, please contact the publisher indicated by the program.

With this virtual reality technique, the user can creat well-controlled motion images with the exact known camera setup and the relative position between the observer and the moving objects. This is a very powerful way of testing and verifying the algorithm developed for motion computation and motion compensation.

2 CREATING IMAGE SEQUENCE WITH BOTH OBSERVER AND OBJECT MOTION

Using the technique described in the previous section, the user can creat image sequence with both observer and object motion. This section describes one example.

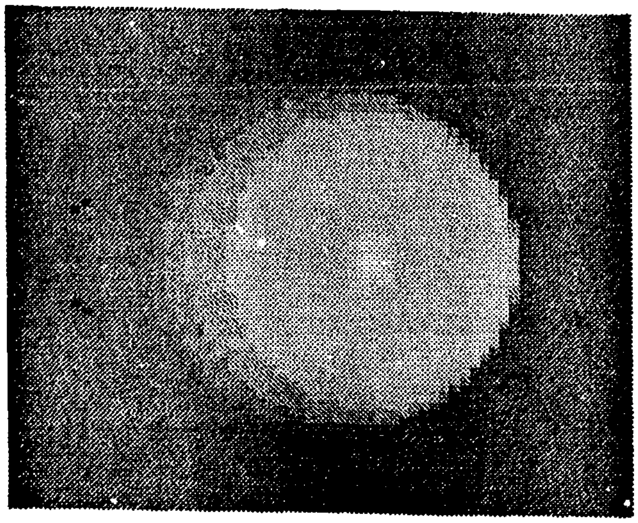
DESCRIPTION: generate squence of moving images including

(1) sphere moves along Z asix with step 2 units;
(2) observer moves along Y axis with step 2 units;
(3) combination of (1) and (2).

The program, BM11.B, for creating 3D virtual environment and the image sequence is given the last chapter of this manual.

3 3D MOTION DETECTION

Using the program GARBOMOTION. three dimensional motion can be computed. The program is the integration of several program modules described in Chapter 3. A source code of the program is given in Chapter 5. The program was documented. With the understanding of the programs in Chapter 3, the use of this program shall not pose new challenge.



Enlarged portion without showing the x-y-z coordinate (movie file is included in the floppy disk and can be played on 486/50Mhz machine or above with SVGA)

Figure 4. A frame of a sequence of images with both observer and sphere motion. The motion parameters and the camera setup were described in the source code listing BM11.B

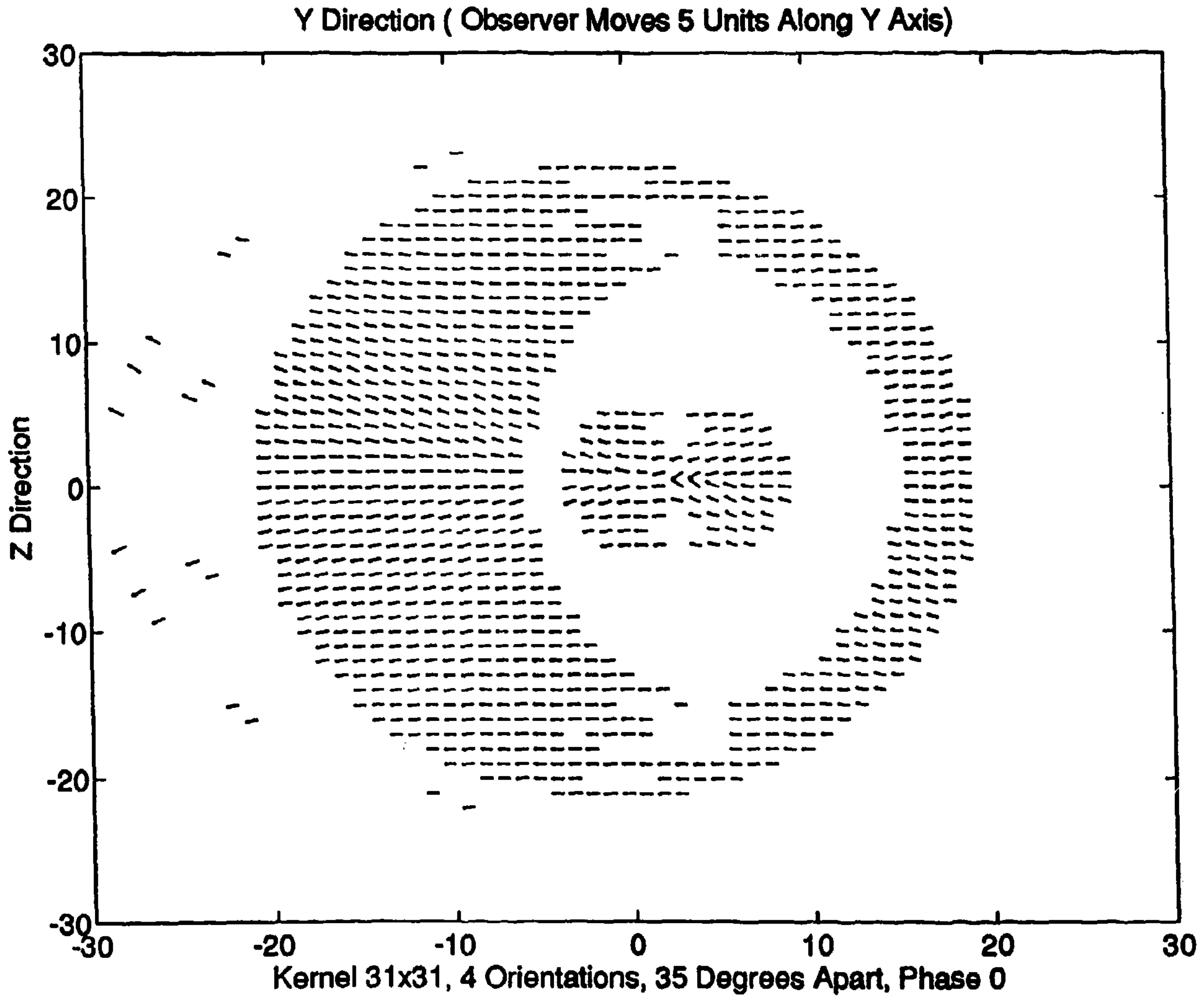


Figure 5. Motion detection result for a sphere image with 4 kernels 45 degree apart.

5 SOFTWARE LISTS

ABSTRACT

This chapter gives the source code listing of all the programs described in this manual. A MS-DOS floopy disk is also provided.

1 BARIMAGE.C

```
/*
 * Copyright (c)  Dr. Hua Li's Lab  1994. All Rights reserved.
 * Purpose: To generate an oriented bar stripe image pattern for verifing
 *          the concept of spatial frequency and orientation bandwidth
 *          selectivities.
 * Image Format: binary image from 0 to 255. The size is ROW x COLUMN;
 * Name: BarImage.c.
 * Compilation: (g)cc -o  outputfilename   BarImage.c -lm
 * Side Effect: No.
 * Input: Spatial frequency, Orientation, and size of image
 * Output: bar image file.
 * Coded by: Xiaohui Meng.
 * Executed Machine: Sun Workstation
 * Last Change: Feb. 24, 1994
 */


#include<string.h>
```

```
#include<stdio.h>
#include<math.h>

BarImg(int row, int column, float omega, float orient, char* filename)
{
FILE *file;
int  i, j, image;
float pi=3.1416, Spatial_Freq_X, Spatial_Freq_Y, Orientation, value;

      Spatial_Freq_X = omega * cos(orient * pi / 180.0);
      Spatial_Freq_Y = omega * sin(orient * pi / 180.0);

      if((file=fopen(filename,"wb"))==NULL ) {
       printf(" Can't open output file for writing");
       exit(-1);
       }


     /*
      *  The bar wave image formulation:
      *     value =   255  if cos(2*pi*(x*wx + y*wy)) > 0.0
      *                          0   otherwise
      *  where wx is the spatial frequency component in u
      *        wy is the spatial frequency component in v
      *  then the value of image is mapped to 0-255
      */

      for ( i = 0;  i < row; i++) {
 for ( j = 0;  j < column; j++) {
              if( cos(2.0*pi*(i*Spatial_Freq_X + j*Spatial_Freq_Y)) > 0.0 )
                 image = 255;
              else
                 image = 0;
                fprintf(file,"%d\t",image);
      }

            fprintf(file,"\n");
}
    fclose(file);
   }
```

2 COSIMAGE.C

```
/*
 *     Copyright (c) Dr. Hua Li's Lab 1994. All Rights Reserved.
 *     Purpose: To generate a oriented cosin  moving image and image derivative.
 *              This pattern is used to test the coordinate system rotation
 *              and image flow.
 *     Name    : CosImage.c.
 *     Compilation: cc -o  outfilename CosImage.c -lm
 *     Output files: (1) orig.img --- original image.
 *                   (2) move.img --- moved image.
 *                   (3) devi.img --- derivative of image.
 *     Coded by: Xiaohui Meng.
 *     Last Change: Feb. 23  1994
 */


#include <stdio.h>
#include <stdlib.h>
#include <math.h>

CosImg(int row, int column, float omega, float orient, float speed, float time)
{
FILE *infile1,*infile2,*infile3;
float value1,value2,value3;
float vx,vy,pi=3.1416;
int i,j;


 /*  Image Pattern Generator
  *  The formula of cosin image is:
  *     I(x,y)= cos(2*pi*omega*(x*cos(theta)+y*sin(theta)-speed*time)).
  *    where   omega is the spatial frequency
  *            theta is the orientation
  *            the product of time and speed defines the phase offset along theta
  *            direction.
  */
         vx=cos(theta*pi/180.0);
         vy=sin(theta*pi/180.0);

         if((infile1=fopen("orig.img","wb"))==NULL)   exit(-1);
```

```
           if((infile2=fopen("devi.img","wb"))==NULL)   exit(-1);
           if((infile3=fopen("move.img","wb"))==NULL)   exit(-1);

          for ( i =-(row-1)/2; i<(row-1)/2+1; i++)
          {
              for (j=-(column-1)/2;j<(column-1)/2+1;j++)
              {
value1 =(int)255.0*(cos(2.0*pi*omega*(vx*(float)i+vy*(float)j-speed*time))+1)/2.0;
value2 =(int)255.0*(cos(2.0*pi*omega*(vx*(float)i+vy*(float)j-speed*(time+1)))+1)/2.0;
value3 =(int)255.0*(cos(2.0*pi*omega*(vx*(float)i+vy*(float)j-speed*(time+2)))+1)/2.0;
              fprintf(infile1,"%d\t",value1);
              fprintf(infile2,"%d\t",(value3-value1)/2.0);
              fprintf(infile3,"%d\t",value2);
              }
             fprintf(infile1,"\n");
             fprintf(infile2,"\n");
             fprintf(infile3,"\n");
             }
          fclose(infile1);
          fclose(infile2);
          fclose(infile3);
}
```

3 COSCONV.C

```
/*
 *     Copyright © Dr. Hua Li's Lab  1994. All Rights reserved.
 *     Purpose: To calculate the image flow from moving oriented cosin images.
 *     Kernel : Gabor real part kernel and image size ROWSIZE x COLUMNSIZE.
 *     Image Format: Binary File
 *     Name   : CosConv.c.
 *     Compilation: cc -o  outputfilename   CosConv.c -lm
 *     Side Effect: Limited by maximum array size
 *     Input: (1) Kernel Modulation Spatial frequency
 *            (2) length  ( length = 1/(sigma*omega))
 *            (3) Kernel Orientation
 *            (4) Thresold of Motion Detection
*           (5) Kernel Phase (0: Gabor Real Part; 90: Gabor Imaginary Part)
*           (6) devi.img: file name of derivative of moving image
*           (7) move.img: file name of moving image
*      Output:(1) u.dat  motion speed along X axis
*           (2) v.dat  motion speed along Y axis
*           (3) t1.dat: gabor response to derivative of moving image
*           (4) x1.dat: gradient gabor response to moving image
*           (5) y1.dat: gradient gabor response to moving image
*      Executed Machine: Sun Sparc Station
*      Coded by: Xiaohui Meng.
*      Last Change: May. 25  1994
*/


#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>

#define   KERNELSIZE   51    /* reserved kernel size */
#define   ROWSIZE      81    /* row size of image */
#define   COLUMNSIZE   81    /* column size of image */

   void main()
   {

   float rpartt[KERNELSIZE][KERNELSIZE],rpartx[KERNELSIZE][KERNELSIZE];
   float rparty[KERNELSIZE][KERNELSIZE],
   buffer1[KERNELSIZE][COLUMNSIZE],buffer2[KERNELSIZE][COLUMNSIZE];
   FILE *infile1,*infile2;
   FILE *out1,*out2,*out3,*out4,*out5;
   int row,col,row1,col1,row2,col2,i,j,index,number,kernsize;
   float vx,vy,omega,theta,pi=3.1416,sigma,orient,length,thresold;
   float image,image1,image2,image3,phase,u,v,uref,vref;
   char   devi_name[20], move_name[20];

           printf("\n Input Spatial Frequency(Cycles/pixel):");
           scanf("%f",&omega);
           printf("\n Input length :");
           scanf("%f",&length);
           printf("\n Input Orientation :");
           scanf("%f",&theta);
```

```
          printf("\n Input thresold :");
          scanf("%f",&thresold);
          printf("\n Input Phase shifting(degrees):");
          scanf("%f",&phase);
          printf("\n Input  Derivative Image File:");
          scanf("%s", devi_name);
          printf("\n Input  Moving Image File:");
          scanf("%s", move_name);

          sigma = 1.0/(length*omega);
          kernsize = (int)ceil(1.0/(omega));
          if(fabs((float)kernsize/2.0-ceil((float)kernsize/2.0))<=0.01)
          kernsize++;
          printf("KernSize=%d\n",kernsize);
          index=(int)(((float)kernsize-1.0)/2.0);


          if((infile1=fopen(devi_name,"r"))==NULL)   exit(-1);
          if((infile2=fopen(move_name,"r"))==NULL)   exit(-1);


  /*------------ Gabor type kernel generator -----------------*/

   vx=omega*cos(theta*pi/180);
          vy=omega*sin(theta*pi/180);
for (i=-index;i<index+1;i++)  {
 for (j=-index;j<index+1;j++)  {
     image=exp(-(float)(i*i+j*j)/(sigma*sigma));
     image1=cos(2.0*pi*(vx*(float)i+vy*(float)j)+phase*pi/180.0)*image;
     image2=sin(2.0*pi*(vx*(float)i+vy*(float)j)+phase*pi/180.0)*image;
     rpartt[i+index][j+index]=image1;
     rpartx[i+index][j+index]=-image2*vx*2.0*pi-image1*(float)i*2.0/(sigma*sigma);
     rparty[i+index][j+index]=-image2*vy*2.0*pi-image1*(float)j*2.0/(sigma*sigma);
 }
}

/*-------------- convolution ----------------------*/

      if((out1=fopen("t1.dat","w"))==NULL) exit(0);
      if((out2=fopen("x1.dat","w"))==NULL) exit(0);
      if((out3=fopen("y1.dat","w"))==NULL) exit(0);
      if((out4=fopen("u.dat","w"))==NULL) exit(0);
```

```
    if((out5=fopen("v.dat","w"))==NULL) exit(0);

    for (row=index;row<kernsize;row++)      /* Initialize the buffer */
    {
 for (col=0;col<COLUMNSIZE;col++)
   {
fscanf(infile1,"%f",&buffer1[row][col]);
fscanf(infile2,"%f",&buffer2[row][col]);
buffer1[row-index][col]=buffer1[index][col];
buffer2[row-index][col]=buffer2[index][col];
    }
    }

 for(i=0;i<ROWSIZE;i++)        /* do the convolution */
 {
             for ( j=0;j<COLUMNSIZE;j++)
    {
     image1=0.0;
     image2=0.0;
     image3=0.0;


                for (row1=-index;row1<index+1;row1++)  /* convolution unit */
    {
     for (col1=-index;col1<index+1;col1++)
{
   col2=j+col1;
   if(col2<0) col2=0;
   if(col2>COLUMNSIZE-1) col2=COLUMNSIZE-1;
   image1+=buffer1[row1+index][col2]*rpartt[index+row1][index+col1];
   image2+=buffer2[row1+index][col2]*rpartx[index+row1][index+col1];
   image3+=buffer2[row1+index][col2]*rparty[index+row1][index+col1];
}
    }

        /*
         * do motion speed calculation
         */

              if( fabs(image2) > thresold )
              u = cos(pi*theta/180.0)*cos(pi*theta/180.0)*image1/image2;
              else u = 0.0;
```

```
                if( fabs(image3) > thresold )
                v = sin(pi*theta/180.0)*sin(pi*theta/180.0)*image1/image3;
                else v = 0.0;

          /*    if(uref =0.0)  u = 0.0;
                else u = uref*(vref*vref/(uref*uref+vref*vref));
                if(vref = 0.0 ) v = 0.0;
                else v = vref*(uref*uref/(uref*uref+vref*vref)); */
                fprintf(out1,"%f\t",image1);
                fprintf(out2,"%f\t",image2);
                fprintf(out3,"%f\t",image3);
                fprintf(out4,"%f\t",u);
                fprintf(out5,"%f\t",v);
             }  /* for loop for j  */

    fprintf(out1,"\n");
    fprintf(out2,"\n");
    fprintf(out3,"\n");
             fprintf(out4,"\n");
             fprintf(out5,"\n");

              for (col=0;col<COLUMNSIZE;col++)     /* shifter */
 {
  for (row=0;row<kernsize-1;row++)
    {
     buffer1[row][col]=buffer1[row+1][col];
     buffer2[row][col]=buffer2[row+1][col];
    }
    if (i<ROWSIZE-index-1)
   {
    fscanf(infile1,"%f",&buffer1[kernsize-1][col]);
    fscanf(infile2,"%f",&buffer2[kernsize-1][col]);
    }
 }
 }
fclose(out1);
fclose(out2);
fclose(out3);
         fclose(out4);
         fclose(out5);
         fclose(infile1);
         fclose(infile2);
```

```
}
```

4 CREATGABOR.C

```
/*
 * Copy right © Dr. Li's Laab 1994. All Rights reserved.
 * Purpose: To generate Gabor kernel.
 * Format:  float point
 * Name:    CreateGabor.c
 */

void CreatGabor(omega, theta, type,size, tempt, tempx, tempy)
float omega, theta;
int type, size;
float **tempt, **tempx, **tempy;
{

float omega_x, omega_y, sigma;
float phase = 0.0;   /* Gabor real part as default */
float image, image1, image2;
float pi = 3.141593;
int i, j;

  omega_x = omega*cos(theta*pi/180);
  omega_y = omega*sin(theta*pi/180);
  if(type == 1 ) phase = 90.0;
  else  phase = 0.0;
  sigma= 1.0/(4.0*omega);

  for (i=size;i>=-size;i--)
  {
   for (j=size;j>=-size;j--)
    {
 image=exp(-(float)(i*i+j*j)/(sigma*sigma));
 image1=cos(2.0*pi*(omega_x*(float)i+omega_y*(float)j)+phase*pi/180.0)*image;
 image2=sin(2.0*pi*(omega_x*(float)i+omega_y*(float)j)+phase*pi/180.0)*image;
tempt[size-i][size-j]=image1;
tempx[size-i][size-j]=-image2*omega_x*2.0*pi-image1*(float)i*2.0/(sigma*sigma);
tempy[size-i][size-j]=-image2*omega_y*2.0*pi-image1*(float)j*2.0/(sigma*sigma);
      }
   }
}
```

5 BUFFERINIT.C

```
/*
 *  Copy Right © Dr. Li's Lib. All Rights Reserved
 *  Purpose:      initialize the image buffer to take care of the
 *                image boundary condition for convolution.
 *  Module Name:  BufferInit.c
 *  Required:     file was opened in "RB" mode.
 *  Image Format: 8-bit/pixel binary format.
 *  Coded by:     Xiaohui Meng
 *  Last Change:  Oct. 1994.
 */


void BufferInit(index, kernsize, column, buffer, fp)
int index, kernsize, column;
unsigned char **buffer;
FILE *fp;
{
   int  row;

   for( row = 0; row < kernsize; row++) {
         if( row < index ) {
           fread(buffer[row],sizeof(unsigned char), column, fp);
           rewind(fp);
           }
         else
           fread(buffer[row], sizeof(unsigned char), column, fp);
     }
}
```

6 BUFFERSHIFTER.C

```
/*
 *  Copy Right © Dr. Li's Lib. All Rights Reserved
 *  Purpose:      shift data in image buffer for convolution.
 *  Module Name:  BufferShifter.c
 *  Required:     file was opened in "RB" mode.
 *  Image Format: 8-bit/pixel binary format.
 *  Coded by:     Xiaohui Meng
 *  Last Change:  Oct. 1994.
 */


void BufferShifter(kern_index, kern_size, row_img, col_img, row_loop, buffer, fp)
int kern_index, kern_size;
int row_img, col_img, row_loop;
unsigned char **buffer;
FILE *fp;
{
   int  row, col;

   for( col = 0; col < col_img; col++)
   for( row = 0; row < kern_size-1; row++)
      buffer[row][col] = buffer[row+1][col];

   if( row_loop < row_img - kern_index -1 )
     fread(buffer[kern_size -1],sizeof(unsigned char), col_img, fp);

}
```

7 CONVOLUTIONUNIT.C

```
/*
 *  Copy Right © Dr. Li's Lib. All Rights Reserved.
 *  Purpose:      convolve image with kernel.
 *  Module Name:  ConvolutionUnit.c
 *  Required:     file was opened in "RB" mode.
 *  Image Format: 8-bit/pixel binary format.
 *  Coded by:     Xiaohui Meng
 *  Last Change:  Oct. 1994.
 */


float ConvolutionUnit(kern_index, col_image, col_loop, image_buf, kern_buf)
int kern_index, col_image, col_loop;
unsigned char **image_buf, **kern_buf;
{

  int row1, col1, col2;
  float conv_data= 0.0;
  for (row1 = -kern_index; row1 < kern_index + 1; row1++ )  {
    for (col1 = -kern_index; col1 < kern_index + 1; col1++)  {
      col2 = col_loop + col1;
      if( col2 < 0 ) col2 = 0;  /*judge boundary in column of image buffer */
      if( col2 > col_image - 1)  col2 = col_image - 1;
      conv_data += (float)image_buf[row1+kern_index][col2] *
        kern_buf[kern_index+row1][kern_index+col1];
     }
  }

  return conv_data;
}
```

8 GABORMOTION.C

```
/*
 *     Copyright © Dr. Hua Li's Lab  1994. All Rights reserved.
*     Purpose: To calculate the image flow from moving images.
*     Kernel : Gabor type  kernel(real or imaginary part).
*     Image Format:   Binary image with ROWSIZE x COLUMNSIZE.
*     Name    : Gabor.c.
*     Required: Leastsq.c.
*     Compilation: cc -o  filename   Gabor.c -lm
*     Side Effect: Limited by maximum array size.
*     Input: (1) Kernel Modulation Spatial frequency
*            (2) Starting orientation of kernel
*            (3) Rotating angle each time
*            (4) Numbers of rotating
*            (5) Threshold
*            (6) Phase in kernel. 0: real part; 90: imaginary part.
*            (7) Derivatevi Image File: file name of derivative of moving image
*            (8) Moving Image File: file name of moving image
*     Output:(1) u.dat  motion speed along X axis
*            (2) v.dat  motion speed along Y axis
*            (3) t1.dat: gabor response to derivative of moving image
*            (4) x1.dat: gradient gabor response to moving image
*            (5) y1.dat: gradient gabor response to moving image
*     Executed Machine: Sun Sparc Station
*     Coded by: Xiaohui Meng.
*     Last Change: Mar.18 1994
*/


#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include "Leastsq.c"

#define   KERNELSIZE   51     /* reserved kernel size */
#define   ROWSIZE      61     /* row size of image */
#define   COLUMNSIZE   61     /* column size of image */

   void main()
   {

float rpartt[KERNELSIZE][KERNELSIZE],rpartx[KERNELSIZE][KERNELSIZE];
float rparty[KERNELSIZE][KERNELSIZE],
buffer1[KERNELSIZE][COLUMNSIZE],buffer2[KERNELSIZE][COLUMNSIZE];
```

```
FILE *infile1,*infile2,*infile3;
FILE *out1,*out2,*out3;
int row,col,row1,col1,row2,col2,i,j,index,number,kernsize;
float vx,vy,omega,theta,pi=3.1415926535,sigma,angle_start,angle_tune,orient_num;
float thresold;
float image,image1,image2,image3,phase;
   char buffer_a[15];
   char buffer_b[15];
   char buffer_c[15];
   char buffer_d[15];
   char *string1=".dat";
   char *string2="t\0";
   char *string3="x\0";
   char *string4="y\0";
   char  devi_name[20], move_name[20];

          printf("\n Input Spatial Frequency(Cycles/pixel):");
          scanf("%f",&omega);
          printf("\n Input Initial Angles(degrees):");
          scanf("%f",&angle_start);
          printf("\n Input Tuning Angles(degrees):");
          scanf("%f",&angle_tune);
          printf("\n Input Orientation Numbers :");
          scanf("%f",&orient_num);
          printf("\n Input Threshhold :");
          scanf("%f",&thresold);
          printf("\n Input Phase shifting(degrees):");
          scanf("%f",&phase);
          printf("\n Input  Derivative Image File:");
          scanf("%s", devi_name);
          printf("\n Input  Moving Image File:");
          scanf("%s", move_name);

    number=0;         /* number of orientations */
          sigma= 1.0/(8.0*omega);   /* set length = 4.0 */
          kernsize = (int)ceil(1.0/(omega));
          if(fabs((float)kernsize/2.0-ceil((float)kernsize/2.0))<=0.01)
          kernsize++;
          printf("KernSize=%d\n",kernsize);
          index=(int)(((float)kernsize-1.0)/2.0);
          if((infile1=fopen(devi_name,"r"))==NULL)   exit(-1);
          if((infile2=fopen(move_name,"r"))==NULL)   exit(-1);


    for(theta=angle_start;theta<=angle_start+(orient_num1.0)*angle_tune+1.0;
          theta+=angle_tune)  {
               number+=1;
               sprintf(buffer_d,"%d",number);
               strcpy(buffer_a,string2);
               strcpy(buffer_b,string3);
               strcpy(buffer_c,string4);
               strcat(buffer_a,buffer_d);
               strcat(buffer_b,buffer_d);
               strcat(buffer_c,buffer_d);
               strcat(buffer_a,string1);
               strcat(buffer_b,string1);
               strcat(buffer_c,string1);



   /*------------Gabor Type Kernal Generator ----------
     infile3=fopen("kern.dat","w");

    vx=omega*cos(theta*pi/180);
    vy=omega*sin(theta*pi/180);
 for (i=index;i>=-index;i--)
 {
  for (j=index;j>=-index;j--)
  {

    image=exp(-(float)(i*i+j*j)/(sigma*sigma));
    image1=cos(2.0*pi*(vx*(float)i+vy*(float)j)+phase*pi/180.0)*image;
    image2=sin(2.0*pi*(vx*(float)i+vy*(float)j)+phase*pi/180.0)*image;
    rpartt[index-i][index-j]=image1;
    rpartx[index-i][index-j]=-image2*vx*2.0*pi-image1*(float)i*2.0/(sigma*sigma);
    rparty[index-i][index-j]=-image2*vy*2.0*pi-image1*(float)j*2.0/(sigma*sigma);
                fprintf(infile3,"%f\t",image1);
  }
 fprintf(infile3,"\n");
             }
 /*------------- convolution ----------------------*/
     if((out1=fopen(buffer_a,"w"))==NULL) exit(0);
     if((out2=fopen(buffer_b,"w"))==NULL) exit(0);
     if((out3=fopen(buffer_c,"w"))==NULL) exit(0);

     for (row=index;row<kernsize;row++)       /* Initialize the buffer */
     {
 for (col=0;col<COLUMNSIZE;col++)
   {
fscanf(infile1,"%f",&buffer1[row][col]);
fscanf(infile2,"%f",&buffer2[row][col]);
buffer1[row-index][col]=buffer1[index][col];
buffer2[row-index][col]=buffer2[index][col];

     }
     }

 for(i=0;i<ROWSIZE;i++)         /* do the convolution */
 {
          for ( j=0;j<COLUMNSIZE;j++)
     {
      image1=0.0;
      image2=0.0;
      image3=0.0;
for (row1=-index;row1<index+1;row1++)  { /* convolution unit */
     for (col1=-index;col1<index+1;col1++)  {
   col2=j+col1;
   if(col2<0) col2=0;
   if(col2>COLUMNSIZE-1) col2=COLUMNSIZE-1;
   image1+=buffer1[row1+index][col2]*rpartt[index+row1][index+col1];
   image2+=buffer2[row1+index][col2]*rpartx[index+row1][index+col1];
   image3+=buffer2[row1+index][col2]*rparty[index+row1][index+col1];
}
     }
               fprintf(out1,"%f\t",image1);
               fprintf(out2,"%f\t",image2);
               fprintf(out3,"%f\t",image3);
          }  /* for loop for j  */

      fprintf(out1,"\n");
      fprintf(out2,"\n");
      fprintf(out3,"\n");

 for (col=0;col<COLUMNSIZE;col++)     /* shifter */
   {
    for (row=0;row<kernsize-1;row++)
      {
       buffer1[row][col]=buffer1[row+1][col];
       buffer2[row][col]=buffer2[row+1][col];
      }
      if (i<ROWSIZE-index-1)
     {
      fscanf(infile1,"%f",&buffer1[kernsize-1][col]);
      fscanf(infile2,"%f",&buffer2[kernsize-1][col]);
      }
   }
   }
  rewind(infile1);
  rewind(infile2);
  fclose(out1);
  fclose(out2);
  fclose(out3);
   }

    fclose(infile1);
    fclose(infile2);

  /*------ Least Square Estimation ------*/

  if ( orient_num == 2.0 )  solve2(ROWSIZE,COLUMNSIZE,thresold);
    else
       solve4(ROWSIZE,COLUMNSIZE,thresold);

    }
```

9 LEASTSQ.C

```
/*
 * This program is to minimize the motion error by using
 *  least-square-estimation.  Two functions are included:
 *  (1) solve4(): function for rotating 4 orientations of Gabor type function
 *  (2) solve2(): function for rotating 2 orientations of Gabor type function.
 *  required:  ddd.dat, the difference of first frame and second frame of images
 */



#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>


solve4(ROW,COLUMN, thresold)
int ROW,COLUMN;
float thresold;
 {
      int i,j,k,index;
      float yt[5],yx[5],yy[5];
      float u,v,ball;
      float  num1,num2,num3,num4,num5;
      FILE *ft1,*ft2,*ft3,*ft4,*ft5;
      FILE *fx1,*fx2,*fx3,*fx4,*fx5;
      FILE *fy1,*fy2,*fy3,*fy4,*fy5;
      FILE *out1,*out2,*out3,*out4;



      if((fx1=fopen("x1.dat","r"))==NULL)   exit(0);
      if((fx2=fopen("x2.dat","r"))==NULL)  exit(0);
      if((fx3=fopen("x3.dat","r"))==NULL)  exit(0);
      if((fx4=fopen("x4.dat","r"))==NULL)  exit(0);

      if((fy1=fopen("y1.dat","r"))==NULL)   exit(0);
      if((fy2=fopen("y2.dat","r"))==NULL)  exit(0);
      if((fy3=fopen("y3.dat","r"))==NULL)  exit(0);
      if((fy4=fopen("y4.dat","r"))==NULL)  exit(0);

      if((ft1=fopen("t1.dat","r"))==NULL)   exit(0);
      if((ft2=fopen("t2.dat","r"))==NULL)  exit(0);
      if((ft3=fopen("t3.dat","r"))==NULL)  exit(0);
```

```
        if((ft4=fopen("t4.dat","r"))==NULL)  exit(0);

        if((out3=fopen("ddd.img","r"))==NULL)    exit(0);
        if((out1=fopen("u.dat","w"))==NULL)       exit(0);
        if((out2=fopen("v.dat","w"))==NULL)       exit(0);
        index=0;

        for(i=0;i<ROW;i++)
        {
for(j=0;j<COLUMN;j++)
   {
    fscanf(fx1,"%f",&yx[0]);
    fscanf(fx2,"%f",&yx[1]);
    fscanf(fx3,"%f",&yx[2]);
    fscanf(fx4,"%f",&yx[3]);

    fscanf(fy1,"%f",&yy[0]);
    fscanf(fy2,"%f",&yy[1]);
    fscanf(fy3,"%f",&yy[2]);
    fscanf(fy4,"%f",&yy[3]);

             fscanf(ft1,"%f",&yt[0]);
             fscanf(ft2,"%f",&yt[1]);
             fscanf(ft3,"%f",&yt[2]);
             fscanf(ft4,"%f",&yt[3]);

             fscanf(out3,"%f",&ball);

    num1=0.0;
    num2=0.0;
    num3=0.0;
             num4=0.0;
             num5=0.0;

       if( ball == 0.0 ) {    /* determine if motion occurs */
            fprintf(out1,"%f\t",0.0);
            fprintf(out2,"%f\t",0.0);
            }
           else {
  for (k=0;k<4;k++)  /* /10.0 for reducing out of range */
   {
     num1+=yx[k]/50.0*yx[k]/50.0;
    num2+=yy[k]/50.0*yy[k]/50.0;
    num3+=yy[k]/50.0*yx[k]/50.0;
           num4+=yt[k]/50.0*yx[k]/50.0;
           num5+=yt[k]/50.0*yy[k]/50.0;
  }

   if(fabs(num1*num2-num3*num3)<= thresold)
   {
    puts(" data overflow ");
    u=0.0;
    v=0.0;
    index+=1;
    printf("\n row=%d, column=%d\n",i,j);
    }
    else
    {
/* Coordinate System would be Consistent with MATLAB Coordinate System */
   e=-(num2*num4-num3*num5)/(num1*num2-num3*num3);
   u=-(num1*num5-num3*num4)/(num1*num2-num3*num3);
     }
  fprintf(out1,"%f\t",u);
  fprintf(out2,"%f\t",v);
      }  /* ball == 0 */
      }
  fprintf(out1,"\n");
  fprintf(out2,"\n");

     }

    printf(" \ntatol dataflow number is %d\n",index);
     remove("t1.dat");
     remove("t2.dat");
     remove("t3.dat");
     remove("t4.dat");
     remove("x1.dat");
     remove("x2.dat");
     remove("x3.dat");
     remove("x4.dat");
     remove("y1.dat");
     remove("y2.dat");
     remove("y3.dat");
     remove("y4.dat");
    }


solve2(ROW,COLUMN,thresold)
int ROW,COLUMN;
float thresold;
{
      int i,j,k,index;
      float yt[2],yx[2],yy[2];
      float  num1,num2,num3,num4,num5,u,v,ball;
      FILE *ft1,*ft2;
      FILE *fx1,*fx2;
      FILE *fy1,*fy2;
      FILE *out1,*out2,*out3;

      if((fx1=fopen("x1.dat","r"))==NULL)   exit(0);
      if((fx2=fopen("x2.dat","r"))==NULL)  exit(0);

      if((fy1=fopen("y1.dat","r"))==NULL)   exit(0);
      if((fy2=fopen("y2.dat","r"))==NULL)  exit(0);

      if((ft1=fopen("t1.dat","r"))==NULL)   exit(0);
      if((ft2=fopen("t2.dat","r"))==NULL)  exit(0);
      if((out3=fopen("ddd.img","r"))==NULL)    exit(0);

      if((out1=fopen("u.dat","w"))==NULL)    exit(0);
      if((out2=fopen("v.dat","w"))==NULL)    exit(0);

      index=0;

      for(i=0;i<ROW;i++)
      {
        fcr(j=0;j<COLUMN;j++)
          {

           fscanf(fx1,"%f",&yx[0]);
           fscanf(fx2,"%f",&yx[1]);

           fscanf(fy1,"%f",&yy[0]);
           fscanf(fy2,"%f",&yy[1]);
          fscanf(ft1,"%f",&yt[0]);
          fscanf(ft2,"%f",&yt[1]);
          fscanf(out3,"%f",&ball);

           num1=0.0;
           num2=0.0;
           num3=0.0;
           num4=0.0;
           num5=0.0;

        if( ball == 0.0 ) {    /* determine if motion occurs */
          fprintf(out1,"%f\t",0.0);
          fprintf(out2,"%f\t",0.0);
          }
         else {
         for (k=0;k<2;k++)  /* /10.0 for reducing out of range */
          {
            num1+=yx[k]/50.0*yx[k]/50.0;
            num2+=yy[k]/50.0*yy[k]/50.0;
            num3+=yy[k]/50.0*yx[k]/50.0;
            num4+=yt[k]/50.0*yx[k]/50.0;
            num5+=yt[k]/50.0*yy[k]/50.0;
          }

           if(fabs(num1*num2-num3*num3)<=thresold)
           {
            puts(" data overflow ");
            u=0.0;
            v=0.0;
            index+=1;
            printf("\n row=%d, column=%d\n",i,j);
            }
            else
            {
/* Coordinate System Transform for U and V Otherwise U and V change each other*/
          v=-(num2*num4-num3*num5)/(num1*num2-num3*num3);
          u=-(num1*num5-num3*num4)/(num1*num2-num3*num3);
             }
          fprintf(out1,"%f\t",u);
          fprintf(out2,"%f\t",v);
       }  /* ball == 0 */
       }
```

```
          fprintf(out1,"\n");
          fprintf(out2,"\n");

     }

     printf(" \ntatol dataflow number is %d\n",index);
     remove("t1.dat");
     remove("t2.dat");
     remove("x1.dat");
     remove("x2.dat");
     remove("y1.dat");
     remove("y2.dat");

    }
```

10 BM11.B

```
studio {
from -30 225 120   //set up position of the camera here and next line
at -10 10 10
up 0 0 1
angle 27.1
res 120 100
aspect 1.2
antialias adaptive
background (0 0 1 )
ambient .8 .8 .8
}
/* light source definition*/
light { type point falloff 1 position  60  120  80 color 25 25 25 }

/* objects defination*/
// for red x-axis //
surface { diff  (.3 .3 .3)   shine 20 .5 .5 .5 }

cone { apex 0 0 0 base 25 0 0 apex_radius 1 base_radius 1 }
cone { apex 25 0 0 base 27 0 0 apex_radius 1 base_radius 5 }
cone { apex 27 0 0 base 35 0 0 apex_radius 5 base_radius 0 }
```

```
sphere { center 35 5 0 radius 1 }
sphere { center 42 5 0 radius 1 }
sphere { center 35 5 10 radius 1 }
sphere { center 42 5 10 radius 1 }
cone { apex 35 5 0 apex_radius 1
base 42 5 10 base_radius 1 }
cone { apex 35 5 10 apex_radius 1
base 42 5 0 base_radius 1 }
//for green y-axis //

surface { diff  (0 1 0)   shine 20 .5 .5 .5 }

cone { apex 0 0 0 base 0 25 0 apex_radius 1 base_radius 1 }
cone { apex 0 25 0 base 0 27 0 apex_radius 1 base_radius 5 }
cone { apex 0 27 0 base 0 35 0 apex_radius 5 base_radius 0 }

sphere { center 5 35 10 radius 1 }
sphere { center 12 35 10 radius 1 }
sphere { center 8.5 35 0 radius 1 }
cone { apex 8.5 35 5 base 5 35 10 apex_radius 1 base_radius 1 }
cone { apex 8.5 35 5 base 12 35 10 apex_radius 1 base_radius 1 }
cone { apex 8.5 35 5 base 8.5 35 0 apex_radius 1 base_radius 1 }

// for blue z-axis //

surface { diff  (0 0 1)   shine 20 .5 .5 .5 }

cone { apex 0 0 0 base 0 0 25 apex_radius 1 base_radius 1 }
cone { apex 0 0 25 base 0 0 27 apex_radius 1 base_radius 5 }
cone { apex 0 0 27 base 0 0 35 apex_radius 5 base_radius 0 }

sphere { center -5 -5 25 radius 1 }
sphere { center -12 -5 25 radius 1 }
sphere { center -5 -5 35 radius 1 }
sphere { center -12 -5 35 radius 1 }
cone { apex -12 -5 35 base -5 -5 25 apex_radius 1 base_radius 1 }
cone { apex -5 -5 35 base -12 -5 35 apex_radius 1 base_radius 1 }
cone { apex -5 -5 25 base -12 -5 25 apex_radius 1 base_radius 1 }


/* Sphere */
surf {
```

```
diff ( .8 .1 .1)
spec .8 .1 .1
shine 30
}
sphere { center 10 60 10 radius 10}   // set up  position of sphere here

/* The end */
```